3-2006

Optimal Periodic Inspection of a Stochastically Degrading System

Timothy B. Booher

OPTIMAL PERIODIC INSPECTION OF A STOCHASTICALLY DEGRADING SYSTEM

THESIS

Timothy B. Booher, Captain, USAF

AFIT/GOR/ENS/06-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GOR/ENS/06-04

OPTIMAL PERIODIC INSPECTION OF A STOCHASTICALLY DEGRADING SYSTEM

THESIS

Presented to the Faculty
Department of Operational Sciences
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
in Partial Fulfillment of the Requirements for the
Degree of Master of Science in Operations Research

Timothy B. Booher, S.B.
Captain, USAF

March 2006

AFIT/GOR/ENS/06-04

OPTIMAL PERIODIC INSPECTION OF A STOCHASTICALLY DEGRADING SYSTEM

Timothy B. Booher, S.B.

Captain, USAF

Approved:

Jeffrey P. Kharoufeh	6 MAR 06
Dr. Jeffrey P. Kharoufeh	Date
Thesis Advisor	
[illegible]	6 MAR 06
Lt Col Mark Abramson, Ph.D.	Date
Committee Member	

AFIT/GOR/ENS/06-04

Abstract

This thesis develops and analyzes a procedure to determine the optimal inspection interval that maximizes the limiting average availability of a stochastically degrading component operating in a randomly evolving environment. The component is inspected periodically, and if the total observed cumulative degradation exceeds a fixed threshold value, the component is instantly replaced with a new, statistically identical component. Degradation is due to a combination of continuous wear caused by the component's random operating environment, as well as damage due to randomly occurring shocks of random magnitude. In order to compute an optimal inspection interval and corresponding limiting average availability, a nonlinear program is formulated and solved using a direct search algorithm in conjunction with numerical Laplace transform inversion. Techniques are developed to significantly decrease the time required to compute the approximate optimal solutions. The mathematical programming formulation and solution techniques are illustrated through a series of increasingly complex example problems.

Acknowledgements

I owe a special debt of gratitude to my advisor, Dr. Kharoufeh, without whom this thesis would not have been possible. Not only did he teach me the requisite material required for this research, he taught me how to conduct academic research that conforms to the highest standards. Lt Col Abramson also deserves special appreciation as his assistance has been invaluable both in the development of the solution methodology and extensive editing of the thesis document. I would also like to thank Kent Mueller, Spirit Meller, Ben Boehm, Gary Krupp and Brent McLamb for assistance in developing potential application areas for this research. Moreover, Fred Gillenwater, Dr. Stephen Baumert, Josh Crouse, John Flory, Tim Pitzer, and Wesely Anderson provided valuable assistance at various times throughout the research process. I am also thankful for the assistance and computational hours provided by Dr. J.O. Miller and Dr. Phil Amburn which enabled me to use the Aeronautical Systems Center Major Shared Resource Center computing assets. My deepest thanks, however, I reserve for my family. First, to my parents who encouraged my academic interest at a young age and have always supported my work with selfless love. My wife and daughter have made great sacrifices for this thesis and provided an inexhaustible source of joy that has sustained my motivation throughout this research. I love them dearly and no words can do justice to the thankfulness they deserve. However, no acknowledgement for any endeavor in which I partake would be complete without thanking Jesus Christ, the author and perfector of my faith. For Him, all I do is intended to be an act of worship. This thesis is no exception. Solo Deo Gloria.

Timothy B. Booher

Table of Contents

Page

List of Figures

List of Tables

OPTIMAL PERIODIC INSPECTION OF A STOCHASTICALLY DEGRADING SYSTEM

1. Introduction

All systems experience gradual deterioration over time until they must ultimately be repaired or replaced. In order to maintain these systems, engineers and analysts must decide when and how to inspect and maintain them. The study of optimal maintenance planning provides a quantitative basis for maintenance decisions and is of interest to industrial, governmental, and military organizations. In most typical scenarios, optimal maintenance is concerned with maintaining a system in a manner that maximizes some measure or benefit, or minimizes the long-run average cost. Models for optimal maintenance planning often employ probability and stochastic process theory in an attempt to model realistic complexities which are inherent in many components, sub-systems, and systems.

Until recently, many optimal maintenance models have lacked sufficient detail to account for the complex interactions of degradation mechanisms that determine component lifetime distributions. Consequently, many of the potential gains of optimal maintenance models remain untapped. While the models considered in this thesis are general, and therefore apply to a wide array of scenarios, the application area of focus for this research is the United States Air Force's maintenance policies.

The United States Air Force maintains equipment ranging from multi-billion dollar weapons platforms, such as the B-2 stealth bomber, to the most mundane tools and facilities. Optimizing the inspection intervals for Air Force equipment is a fertile area with the potential to reduce costs dramatically while simultaneously increasing operational availability. Optimal maintenance theory is especially needed in the current environment where senior Air Force leadership has challenged all Air Force maintenance processes to increase availability by an average of 20% while achieving a simultaneous 10% decrease in cost by the year 2011. This research provides a previously non-existent capability to link availability and cost together in a single model producing an approximate optimum inter-inspection interval that maximizes the limiting average availability while satisfying a prespecified budget.

Current U.S. Air Force policies mandate the implementation of *reliability-centered maintenance*, which specifies that all Air Force organizations maintaining equipment must have a plan to consider the reliability measures of the components for which they are responsible to maintain. Most often, reliability engineers in these organizations use statistical data, often collected from an analogous system, to fit a Weibull distribution to estimate the necessary reliability measures which are then used to determine inter-inspection intervals. The cost of this maintenance policy is determined at some later time, and an iterative refinement process ensues before converging on a compromise between cost and availability. Moreover, the current methods used to determine if existing inspection-interval lengths can be extended are even more simplistic. An informal survey found that the most common method used by Air Force depot engineers to extend inspection intervals is to set aside several systems for inspection at the increased interval of interest. When inspected after this prolonged period of time, the systems were closely inspected to see *a posteriori* if any damage was outside acceptable limits.

For currently fielded systems, system managers acknowledge that 90% of the reliability characteristics of a system are set by the time 10% of the program dollars are expended. Therefore, it is critical that analytical tools are available as early as possible in the acquisition process. While the assumptions behind any model (mathematical or otherwise) cannot be completely validated, an educated initial value for an appropriate inter-inspection interval that maximizes availability would be a useful starting point for engineers. An application area of particular interest to the Air Force is to use this starting point when determining inspection intervals during the logistics supportability analysis required before a system can be fielded.

An accurate understanding of a component's reliability has special meaning in a military context. Military applications require equipment to operate in the harshest conditions and must have reliability measures sufficient to both protect lives and accomplish the required mission. In order to illustrate and further motivate the relevance of this thesis, an application concerning the maintenance of Radar Absorbing Material (RAM) is next presented.

Low observable (LO) technologies make weapons systems difficult to detect, track, and engage. These weapons systems, termed *stealth* assets, are vital to the future of the Air Force and have been combat-tested in several operations including Operation Desert Storm, Operation Allied Force and Operation Iraqi Freedom. Stealth aircraft are particularly valuable against high-value targets, which are often either out of range of conventional aircraft or too heavily defended for conventional aircraft to strike. This critical role in combat operations will increase in importance in the coming years, as Russian-built surface-to-air missile systems, such as the SA-10, SA-12, SA-15, and SA-20 appear on the open market. These integrated air defense systems are highly mobile, networked, and possess an entire suite of anti-aircraft engagement mechanisms. Because of these capabilities, surface-to-air missile strikes, and counter-ground jamming, combined with conventional strike aircraft, will not be adequate to gain and maintain air supremacy. Clearly, low observable technology will play a vital role in both low and high-intensity operations.

While LO technology mitigates detection from many sensors, such as heat seekers (infrared), sound detectors, and even the human eye, the ability to reduce the radar cross section (RCS) is a key component of stealth. There are two approaches to reduce the passive radar cross section: shaping to minimize backscatter, and using RAM coating for energy absorption and cancelation. While both these mechanisms have proven effective in reducing RCS, the maintenance requirement associated with RAM is often an area of concern.

Not only are these materials extremely sensitive to the ambient environment, they must maintain precise tolerances for an entire array of electrical properties in order for the entire system to effectively reduce a system's RCS. Therefore, the effect of environmental exposure has a direct impact on aircraft RCS and consequently aircraft survivability. This concern persists even in the most benign environments, as optimal flying conditions have been shown to induce noticeable wear on RAM [137]. Sunny weather, rain, and sandy environments all cause RAM to degrade linearly at different rates. The B-2, for example, must be stored in a climate-controlled hangar to mitigate RAM near-field reflectivity loss. Near-field reflectivity is an important measure of RAM performance and can be measured at a specific location. Measuring near-field reflectivity per periodic inspections by maintenance personnel is more cost effective than measuring the overall RCS degradation, which requires an entire aircraft to be tested in a specialized facility.

Environment-dependent wear alone cannot adequately characterize the RAM deterioration process. Shocks also provide an extremely important degradation mechanism. For example, in-air refueling stops are short periods of time that can cause rapid damage when the refueling aircraft's boom scratches the sensitive LO surfaces. Other shocks include bird strikes and maintenance personnel touching the RAM surface. The linear wear rates for near-field reflectivity loss are easily measured experimentally. Shock damage is also measurable from individual maintenance records, and shock arrival rates are readily obtained from historical data in current maintenance databases.

Due to the deterioration process described above, RAM must be routinely inspected and replaced. Since RAM deterioration can only be measured when inspected, failures are said to be *hidden* or *non self-announcing*. The inspection interval is very sensitive with regard to cost and operational availability since in order to inspect RAM deterioration, an aircraft must be removed from operational use. An overly small inspection interval could be operationally and financially expensive, but too lax an inspection interval could potentially result in a compromise of aircraft survivability and risk of loss of an extremely costly asset.

A stochastic model developed by Kharoufeh et al. [67] is well-suited for the RAM degradation process and presents reliability and availability measures for a component subject to degradation from a simultaneous combination of linear wear from a random environment and from random shocks. The combination of these two degradation mechanisms gives this model considerable flexibility in addressing a wide-range of applications as many systems are maintained by inspections conducted according to a fixed, deterministic interval. Concerning the previous example, with correctly identified parameters for the shock and wear, and an appropriate optimization model, it is possible to compute an optimal inter-inspection interval that maximizes the limiting average availability of a stealth weapons system.

Availability is, in the most general sense, the proportion of time that a component or system is available to be used for its intended purpose. Naturally, availability is an important measure of concern in many applications and is generally desired to be as high as possible. While the model of Kharoufeh et al. [67] has wide applicability and provides a mechanism to understand the impact of a particular selection of the inter-inspection interval on reliability and availability measures, no previous research has developed a methodology to maximize the limiting average availability through selection of the inter-inspection interval for a model that considers degradation due to environment-induced wear and random shocks.

1.1 Problem Definition and Methodology

This thesis will develop an appropriate cost function and formal nonlinear program to maximize the limiting average availability of a component subject to linear wear and random shocks. The decision variable of interest is the fixed length of the inter-inspection interval. An appropriate optimization technique will be selected and implemented through an accessible set of computer codes. Finally, several test cases will be constructed to illustrate the optimization technique in practice.

However, in order for the methodology to be useful in an industrial or military setting, it must be able to obtain an answer in a reasonable amount of time. Therefore, the research objectives of this thesis are: (1) to construct the necessary optimization methodology to maximize availability, and (2) to produce this answer in the least amount of time possible.

To accomplish these two objectives, the stochastic degradation model developed by Kharoufeh et al. [67] will be reviewed. Next, a cost function will be developed which considers the long-run cost per cycle due to component replacement, inspections, and downtime. The objective function of [67] and the associated cost function combine to form a nonlinear program. However, because the representation for the limiting average availability is provided only in the form of a Laplace transform, real-domain derivative information is not available, and a solution requires the use of Laplace transform inversion techniques. For this reason, many standard optimization techniques are inadequate; thus derivative-free algorithms (particularly pattern search) are considered.

Since the objective function producing the limiting average availability is computationally intensive, a study was conducted to reduce the run times of its most frequently called subfunctions: the matrix exponential and the Laplace transform inversion algorithm. Four methods to compute the matrix exponential were studied and a Laplace transform inversion algorithm was selected. A complexity analysis was conducted on the direct computation of the limiting average availability and cost computations. This leads to the development of a method that dramatically reduces run times as the number of environmental states and system maximum lifetime increase. All computational methods and the overall optimization problem are illustrated through five test-cases.

1.2 Thesis Outline

The next chapter includes a brief survey of the vast literature in the field of optimal maintenance. Starting with the first transition of ideas from other disciplines into an optimal maintenance context, the review surveys three frequently used stochastic deterioration mechanisms and the literature on optimal replacement. The literature reviewed will describe the history and variations of currently used optimal maintenance models, with an emphasis on the historical thread that develops into the model examined in this thesis. The goal of this survey is to provide the reader the necessary background to understand the particular contributions of the thesis in their historical context. This chapter highlights the need for an implementable optimization methodology to determine the optimal period of time to wait between inspections to maximize limiting average availability for a given budget constraint and associated cost function.

The formal notation and mathematical model are developed in chapter 3. The first section describes the notation and formulation of the stochastic degradation model developed by Kharoufeh et al. [67], which presents reliability and availability measures for a system that degrades according to a simultaneous combination of linear wear and random shocks. The second section describes the development of a cost function that considers the long-run cost per cycle of each inspection. In the third section, a formal, constrained nonlinear programming formulation is presented.

Chapter 4 discusses the solution methodology used to solve the nonlinear program presented in chapter 3. This chapter starts with a discussion of potential solution methodologies and their degree of applicability to the nonlinear program in question. In particular, non-derivative-based search procedures are considered with pattern search chosen as the best overall optimization procedure. In order to decrease the run time of the optimization procedure, algorithmic refinements are made to the matrix exponential and the Laplace transform inversion algorithm. The chapter also presents the development of a numerical method to dramatically reduce run times for the chosen methodology.

Chapter 5 presents the results of several numerical experiments. The limiting average availability is computed using pattern search methods with four different methods to calculate the matrix exponential. The result from this experiment is the most suitable matrix exponential method, which is then used in a second experiment to isolate the gains provided by a new method developed to decrease the computational run time. The thesis is concluded in chapter 6 by summarizing the main results and discussing the contributions of this research, and making some final recommendations as well as suggestions for future research directions.

2. Review of the Literature

This chapter provides an overview of the literature related to the area of operations research termed *optimal maintenance*, with an emphasis on optimal replacement models and availability optimization. Optimal maintenance is a well-established field within operations research concerned with maintaining a component in a manner that maximizes profit or minimizes cost.

This review begins with a discussion of motivations for implementing optimal maintenance and its general methodology followed by a discussion of the history of optimal maintenance models. Since the foundation of any maintenance model relies on the underlying deterioration process and failure behavior of the component, three common degradation models are reviewed: shock models, wear models and compound models that consider degradation from both shock and wear. After covering degradation models, optimal replacement models for both preventive and corrective maintenance are reviewed. That section concludes with a discussion of recent models for availability analysis of complex components.

All components degrade over time and actions must be taken to keep equipment operating for its intended purpose. These actions include both preventive actions undertaken at regular intervals to prevent an unacceptable loss of performance and corrective actions taken to restore a failed component to an operational state. In both cases, maintenance actions can be accomplished at fixed intervals or based on conditions concerning the component of interest.

In this review, a component describes any system or subsystem that is modeled and evaluated as a single entity. Thus, a system that consists of multiple components, yet works towards a single objective can be considered a component in order to reflect the aggregate system-level behavior. Thus, the detailed interactions between the various subcomponents can be neglected.

In many industries, costs associated with the allocation of resources to maintenance are very significant, and maintenance personnel comprise a significant number of the total workforce. The potential impact of maintenance at the level of operations and logistics is considerable, and the financial implications of maintenance can be substantial. Therefore, models are required to gain greater insight into maintenance processes in order to better set inspection intervals and understand process dynamics.

Optimal maintenance models have diverse application areas, including crack growth ([123] and [100]), airframes [112], offshore structures, coastal flood barriers subject to erosion [125], and many other industrial and military settings. While extensive effort has been devoted to developing simulation models, as surveyed by Czajkiewicz [49], simulation modeling does not give insight into the underlying structure of maintenance problems and may provide inaccurate answers, while simultaneously shielding the source of these errors. Further, even with recent advances in processor speed, simulations can take long periods of time to compute an approximate answer, as compared to the seconds required to compute an exact answer by analytical methods.

2.1 The History of Optimal Maintenance

The beginning of modern optimal maintenance is closely correlated to the beginning of operations research in general which was developed during the second World War. The foundations of the first optimal maintenance models came from actuarial research performed in Switzerland in the early 20th century. This research focused on determining the number of annual accessions required to maintain a finite body of policyholders [82]. Lotka [82] adapted this research through the use of the theory of renewals to address industrial replacement problems. Lotka's research was followed by more applications of renewal theory and actuarial models to structural reliability and fatigue failure applications. The crossover from other disciplines into optimal maintenance theory continued, as theory originally developed for population analysis and problems in genetics was applied to a number of industrial problems.

During the second World War, the fields of reliability and optimal maintenance were studied with renewed interest as both the industrial economy and academic community focused on assisting the war effort. During this time, work by Weibull [142] focused on approximating probability distributions to model the failure mechanics of materials and introduced the well-known *Weibull* distribution for use in modeling component lifetimes. Shortly thereafter, Davis [50] demonstrated that the exponential distribution worked well for modeling many components. Today, the exponential distribution remains the distribution of choice for many maintenance modeling problems, as it has been demonstrated to aptly model the real-world. It is also the only distribution to possess the memoryless (or Markov) property, which allows for easy aggregation of failure rates of subcomponents to determine the failure rate of the overall component.

Maintenance models increased dramatically in complexity in the 1960s as the study of preventive maintenance as a research discipline began to appear in the literature. Researchers such as Barlow [28], Proschan [29], Jorgenson [65], McCall [87], Radner [65], and Hunter [28] contributed extensively to the early development of maintenance optimization models. Applications in this era focused on systems with potentially catastrophic failures, such as nuclear power plants and optimal maintenance for intercontinental ballistic missiles [65]. The increased complexity of many of these models required the relaxation of the exponential distribution, and research expanded to include semi-markov processes [122].

Current research uses stochastic processes to describe the failure-generating mechanisms of optimal maintenance models. Renewal theory remains the prevalent method used to model stochastic failure processes. Despite the broadening scope of modern research to include more complex components, many studies have shown that renewal theory is capable of accommodating these complexities. The following sections will survey the current literature in optimal maintenance starting with a review of the underlying deterioration process and failure behavior of components in optimal maintenance models.

2.2 Degradation Mechanisms

Stochastic degradation models are mathematical models which attempt to describe a component's deterioration over time. There are two primary mechanisms presented in the literature for modeling a component's aging process over time: the use of existing lifetime distributions or mathematical modeling of the physical dynamics that cause the failure behavior. Lai and Xie [75] provide an overview of model aging and dependence characteristics in reliability and survival analysis. Existing distributions fall into the broad categories classified by how their failure rate changes over time: increasing, decreasing, or constant. Since the foundation of any maintenance model relies on the underlying degradation process of the system or component under consideration, the following section will review three primary degradation models actively used in current research: shock models, wear models and compound damage models.

Shock models are effective in describing a component whose degradation is the result of a collection of distinct stresses, termed *shocks*, applied at discrete points in time. The literature provides a wide variety of methods to describe the frequency of occurrence, and magnitude of damage caused by shocks. Most commonly, however, probability distributions are used to model damage magnitude, and shocks arrive according to a Poisson process. There are two broad areas of the failure mechanisms into which shock models can be classified: cumulative damage shock models and maximum shock models. Cumulative damage models consider a failure to occur when the sum of the effects of random shocks over time exceeds a particular threshold value. In contrast, components described by maximum shock models fail when the magnitude of a single shock exceeds a particular threshold value.

Early Poisson shock models are presented by Esary et al. [54] and Ross [110]. Shanthikumar and Sumita [115] present a more complex shock model in which shock magnitudes are correlated with the length of the interval to the next shock. Two years later, Sumita and Shanthikumar [129] extended their previous model with the development of a cumulative shock model. Rangan and Sarada [108] present the earliest paper found during this survey that uses a non-homogeneous Poisson process to model the shock arrival process.

Mallor and Santos [84] classify general shock models used for component reliability. They also present application areas, noting that extreme and cumulative shock models may be appropriate descriptions for the fracture of brittle materials and for damage due to earthquakes or volcanic activity. They employ Laplace transforms to provide the distribution function of the component failure time and its mean value. Råde [107] presented a shock model with a finite number of identical components, each receiving shocks arriving occurring to a Poisson process. He used this model to calculate the time-dependent probability to failure for each identical component. Nakagawa [96] developed a replacement policy for Råde's system of identical components, which exchanges a given component before failure if the total number of failed components is more than a fixed number, n, and which replaces the system if all n components have failed. Nakagawa determined the optimal number, n, that minimizes the expected cost. Igaki et al. [62] consider a state-dependent shock model influenced by an external system, in which both the interarrival time and the magnitude of the shock are determined by a Markov process. Ebrahimi [53] presents a model subject to shocks governed by a Poisson process, where damage accumulates additively and the component fails if the total damage exceeds a certain capacity or threshold. His results allow for the comparison of two random processes by stochastic ordering.

While shock models can adequately model many components, they require the application of stresses at finite intervals of time. *Wear* models, however, can model the continuous effect of deterioration over time. Wear models were introduced historically later than shock models by Esary et al. [54], which proved that if the lifetime distribution $H(t)$ of a device is subject to shocks governed by a Poisson process with the probability P_k of surviving k shocks, and if the discrete distribution is increasing failure rate average (IFRA), then $H(t)$ is also IFRA. The class of IFRA life distributions plays a fundamental role in reliability theory. A distribution function H with survival function $\bar{H} = 1 - H$ is said to have an increasing failure rate average if $H(0) = 0$ and $\bar{H}^{-1}(t)$ is decreasing for $t > 0$. Singpurwalla [119] provides a survey of models that use the effects of a common environment as a basis for generating dependent lifetimes for components in a system.

Çinlar [45] introduced a Markov additive process (MAP) to describe the failure mechanism due to wear. Markov-additive processes (MAPs) are a class of Markov processes which have important applications. A MAP $\{(X(t), J(t)) : t \geq 0\}$ is a bivariate Markov process whose transition probability measure is translation invariant in the additive component $X(t)$. Here $X(t)$ is any independent CTMC from which $J(t)$ is an additive functional. Çinlar uses the unique structure of the MAP to prove that, given a gamma process (a stochastic process with independent increments) with a shape parameter that is a function of Brownian motion, the resulting lifetime is distributed according to the Weibull distribution. Kharoufeh [66] presents a compact transform expression for the failure distribution for wear processes of a component degrading according to a Markovian environment inducing state-dependent continuous linear wear. He accomplishes this by using the properties of a MAP and assuming the wear process to be temporally homogeneous and that the environmental process has a finite state space.

Though less discussed when compared to individual shock and wear models, compound damage models consider the combined effect of wear and shocks on the lifetime of a component. Only the model of Kharoufeh et al. [67] considers the deteriorative effect of linear wear and random shocks simultaneously. Çinlar [45] derives the failure time distributions and associated properties for models subject to continuous wear and shocks using a compound Poisson process. However, his results are complex and do not provide a computation of availability measures. Klutke et al. [71] presents availability measures for a component with random inspection times and wear rates. Klutke and Yang [70] study components that deteriorate due to both shocks and graceful degradation with periodic inspections, but do not consider the effect of wear and shocks occurring simultaneously. Using regenerative arguments, they derive an expression for the limiting average availability. Kiessler et al. [68] studied inspected components with non-self-announcing failures, where the rate of deterioration is governed by a Markov chain. They assumed environmental exposure changes randomly over time and employed a Fourier series expansion to compute the lifetime distribution and availability when the component is inspected according to a periodic inspection policy.

Taken together, the shock, wear and compound damage models discussed herein provide a number of mechanisms useful for modeling the deterioration of a variety of systems. However degradation models provide only a means to understand when components will fail and do not provide a mechanism to maintain them. To that end, the next section will survey optimal maintenance primarily in the context of optimal replacement models which maintain a system through replacing failed components according to a wide range of policies.

2.3 Optimal Replacement Models

Optimal maintenance models have received a great deal of interest in the last forty years, and numerous surveys of optimal maintenance have been contributed to the operations research literature. The first was completed by McCall [87] in 1965, which covered early preventive maintenance models and classified various preventive maintenance policies by their common characteristics. That same year, the foundational text of Barlow and Proschan [29] was published. Pierskalla and Voelker [104] and Osaki and Nakagawa [102] provided an update by surveying the optimal maintenance research accomplished in the 11 years elapsed since the publication of [87]. These papers were followed by the survey of Sherif and Smith [117] in 1981 that categorized 524 different optimal maintenance papers. This survey was followed eight years later by Valdez-Flores and Feldman [131], who covered optimal maintenance research since [104]. More recent surveys were presented by Cho and Parlar [43] in 1991, Murdock [93] in 1995, Dekker [51] in 1996 and Wang [138] in 2002.

Optimal maintenance models can be classified by the length of planning horizon for the problem, the degree of effectiveness of each maintenance action, the optimization criteria, and the nature of the degradation process and state space that characterize the component [131]. A key distinguishing feature among these models is the degree to which a component is repaired. A particular advantage of perfect repair models is that they are able to take advantage of renewal theory, as presented in [30] and [111]. This section will briefly survey non-replacement repair models and will then emphasize and survey replacement models presented in the literature. Replacement or renewal refers to maintenance models in which a given component is placed into a state after repair equal to the condition the component was in at the start of the process.

The first alternative to replacement or *perfect repair* is termed *minimal repair*. Minimal repair was introduced by Barlow and Hunter [28] and refers to models in which maintenance actions return a component to a state that is stochastically identical to the state just prior to failure. Thus with minimal repair the failure rate is identical before and after a repair operation. Minimal repair is often motivated by considering the increased cost of replacing a component, when compared to that of a simple repair. Several minimal repair models are discussed in the literature [113, 18, 118].

As a combination of perfect and minimal repair, *imperfect repair* is incorporated into models in which maintenance does not renew a component to its original state. First presented by Brown and Proschan [36], imperfect repair is surveyed later by Pham and Wang in [139]. The associated mathematics of imperfect repair is more involved, as standard renewal theory does not apply. Several papers presenting results for imperfect repair are surveyed. Recent research considering imperfect repair includes Bruns [37], who studies a repairable component with Markovian deterioration and imperfect repair options and presents optimal strategies to minimize long-run costs. Biswas et al. [33] model a component which is maintained through a periodic inspections with maintenance accomplished by a combination of imperfect and perfect repair. They combine the maintenance mechanisms by replacing a failed component after a fixed number of imperfect-repairs. Cha et al. [41] compare steady-state availabilities of two different components subject to imperfect repair policies. They make comparisons of the steady-state availability based on imperfect repair policies. Kijima and Nakagawa [69] present replacement policies for a shock model with imperfect preventive maintenance.

Replacement models can be classified as either preventive or corrective. Corrective maintenance considers all maintenance actions performed in response to a component failure. Preventative maintenance is concerned with actions performed to decrease the probability of unplanned failure. In order for preventive maintenance to be viable, the cost of a preventive repair must be less than the cost of repairing a failure.

There are two primary types of preventive maintenance policies: block replacement and age replacement. Under a block replacement policy, maintenance actions are performed at fixed and deterministic time intervals. Under an age replacement preventive maintenance policy, maintenance is initiated when a component has accumulated a certain amount of operational time. The survey papers mentioned before, especially [87] and [104], cover many variations of preventive maintenance models, with an emphasis on single unit components. Research in the area of age replacement policies is primarily concerned with the optimal operational time a component is allowed to accumulate before initiating a maintenance action. Common assumptions to accommodate the use of basic renewal theory include that the time required to replace a failed component is considered negligible and that a component is repaired perfectly (i.e. repaired to a new state). Numerous cost models have been presented to determine the optimal replacement age, most notably by Ascher and Feingold [19], Aven and Bergman [26] and Nachlas [94].

Block replacement is concerned with implementing a policy of maintenance at regularly spaced intervals without regard to the age of the component. Block replacement problems are popular, partially due to the fact that they are often easier to implement mathematically and operationally. Barlow and Proschan [30] proved that a block replacement policy results in less failures than an age replacement policy, even though block replacement requires more replacements and consequently incurs higher cost. Wortman et al. [144] determined that deterministic block replacement is preferable to random replacement. The literature discusses many preventive maintenance variations of block replacement, many with the purpose of presenting policies that decrease the cost of frequent replacement or simplify the mathematical expressions incurred by block replacement. For example, Barlow and Hunter [28] consider a minimal-repair preventive maintenance model with block replacement. Shaked and Zhu [114] summarize and survey various preventive maintenance models that use renewal theory to develop the mathematical framework for block replacement policies.

Models have also been presented that combine preventive and corrective maintenance, such as the work by Nachlas and Cassady [95], who seek to determine an optimal balance between preventive maintenance and corrective maintenance. Some papers present and analyze both corrective and preventive maintenance as potential maintenance strategies, while placing emphasis on a particular recommendation.

While block replacement models suffer from the potential to prematurely replace components and age replacement policies require knowledge of the cumulative operational time for the component under consideration, *inspection models* are applicable when the state of a component is known only during an inspection. Many inspection models use a control limit policy, which determines if a component should be replaced upon inspection if a failure or cumulative degradation level is detected. Control limit policies often use the state of the component at a current inspection to determine the optimal time until the next inspection. The potential disadvantage of inspection policies is that the component under consideration remains in a failed state until the next inspection, and the cost of a component in a down state must be considered.

One of the first inspection models presented was by Luss [83] in 1976, where the state of a component is modeled by several levels of deterioration and the cost of replacement after a failure is higher than before. Zuckerman [146] reviews inspection models and presents an optimal inspection interval and replacement rule with consideration for the associated costs when degradation is due to a Poisson shock process. The research presented by Wortman et al. [144] evaluates two different inspection policies for non-self announcing failures that occur as a result of a Poisson shock process. Abdel-Hameed [10] studies reliability measures when degradation occurs according to a Markov process and inspection intervals are fixed. More recently, Vaurio [136] presents an inspection model with replacements occurring according to an age replacement policy.

Optimal maintenance models can be classified as concerned with the maintenance of either a single unit or of multiple units. Barlow and Proschan [29] introduce many single-unit models and authored the classic textbook used for the mathematical theory of reliability. Nakagawa and Osaki [97] followed this with an optimal replacement model that considers a limited number of spares. Multiple-unit maintenance models often suffer from a state-space explosion problem. While much has been demonstrated concerning multiple unit maintenance models using simulation, recent research uses dynamic programming and other techniques to reduce the complexity incurred by an expanding state size. Ben Ari and Gal [31] use a dynamic programming approach to find an optimal replacement policy for a multi-component system. Flynn et al. [56] present an optimal replacement model using a stochastic dynamic program with the goal of finding the optimal balance between the cost of component replacement and the cost of component failure. Branch-and-bound provides an alternative to dynamic programming for solving multi-component reliability problems, as presented by Chung and Flynn [44].

Availability is commonly used as a performance measure in optimal maintenance models, and is broadly defined as the fraction of time a component is in an operational state over the total time. The optimization of availability is both an active area of current research and a fertile ground for future work. Maintaining a component for optimum availability is a key concern in many military and operational settings. The following is a survey of the optimal maintenance literature concerned with optimizing and measuring availability for stochastically degrading components.

Wortman and Klutke [145] examine the availability of a maintained component, where the rate of deterioration is governed by an random environment. With the objective of exploring the influence of random deterioration on component availability, they provide a result that exposes the relationship between the remaining lifetime, environment, and repairs. Furthermore, the authors develop simple bounds that can be used to choose inspection rates that guarantee a specified level of availability. The principal result requires no specific distributional assumptions.

Early availability models include Ahmed and Schenk [15] and Srinivasan and Ramachandra [126]. In 1978 Ahmed and Schenk [15] used optimal control theory to obtain the optimal policies for a component with variable failure and repair rates, as well as variable maintenance and downtime costs. Their optimization model considered the cost of repair, as well as the downtime cost. They used their method to consider both static and dynamic optimal maintenance policies and note that the optimal maintenance policy is highly dependent on the failure rates of the individual system components. Srinivasan and Ramachandra [126] applied Kalman filtering techniques to reliability smoothing and used these to compute instantaneous availability as a function of time. From this they were able to approximate the sampling interval and the measurement variance for a specific maintenance policy. They also proved that estimates of point availability converge in the steady-state.

Other optimization problems have been presented more recently. Reineke et al. [109] calculate an age replacement time in order to maximize interval availability. They extend this with models that optimize either the limiting availability or the cost rate of complex components. Cassady et al. [39] present a model that approximates the replacement age in order to maximize the average availability using experimental design and regression analysis. More recently Cassady, Pohl, and Jin [40] presented an availability optimization model that applies to a general class of two-state repairable components by determining a set of availability importance measures. Through five examples, they demonstrate that focusing on the reduction of component failures provides greater benefit than increasing the speed of equipment repair. The authors then established a set of three optimization models that capture the trade-offs between improving availability performance and the investments required to achieve that improvement.

This chapter has covered a broad survey of the literature of optimal maintenance relevant to this thesis. First, the history of optimal maintenance was discussed, followed by an overview of the relevant literature of three key degradation measures: degradation due to shocks, degradation due to wear, and degradation due to compound models combining the effects of shocks and wear. Then the defining characteristics of optimal maintenance models were discussed with a primary emphasis on replacement models. Availability was presented as a useful measure to maximize in optimal maintenance models, and several papers were reviewed with associated models that optimize availability. The next chapter will focus on the particular degradation model used in this thesis: a compound model considering wear due to a random environment and shocks arriving according to a Poisson process with a general damage distribution and an associated cost function that considers costs of replacement, downtime, and inspections.

3. Mathematical Model and Dynamics

In order to optimize the limiting average availability by selecting an appropriate inter-inspection interval, it is necessary to obtain explicit representations for both the availability and cost measures. The former is provided by Kharoufeh et al. [67], who derive the lifetime distribution as well as the limiting average availability for a component subject to linear wear and random shocks. In their paper, the authors explicitly derive the system's lifetime distribution and mean time to failure, as well as an equation for the limiting average availability. Due to the relevance of their results to this research, their main results are briefly reviewed.

3.1 Stochastic Degradation Model

Consider a component evolving in a random environment that transitions according to a Markov chain in continuous time. Degradation is due to the combination of two damage mechanisms: environment-dependent linear wear and random shocks. Degradation accumulates monotonically until an inspection detects cumulative damage in excess of a fixed threshold, at which point the component is instantly replaced with one in new condition. The maintenance policy is to inspect the cumulative degradation of the system periodically at a fixed interval of length τ and to replace the component if degradation is found to be beyond a fixed threshold value.

Assume the environment is composed of a finite set of states, each corresponding to a particular constant linear wear rate. All wear rates are assumed to be positive, which implies that every environmental state must continuously induce some finite linear wear per unit time on the component. The second degradation mechanism considers random shocks with a random arrival rate and a single general distribution for the damage caused by each shock. Shocks are assumed to arrive according to a Poisson process with rate parameter, λ where λ is a positive scalar. The overall effect of each shock is considered to be small relative to the degradation process, but the cumulative effect of many shocks can be significant. It is also possible that shocks, unlike wear, may cause zero damage. The wear process is assumed to be stochastically independent of the shock process.

Component inspections are assumed to be instantaneous and *perfect*, in that there is no measurement error in determining the state of the system. Failures are *hidden* in that the state of the system can only be discerned during an inspection. A *soft failure* occurs when the total cumulative degradation exceeds some fixed threshold value, x. The time at which this threshold is crossed can be considered the component's useful service life. Immediately upon inspection, a failed component is instantly replaced by a new identical component. Partial repairs are not allowed.

The environment-dependent rate of wear is modulated by a continuous-time, stationary, ergodic continuous-time Markov chain (CTMC), $\mathscr{Z}$, where $\{Z_t : t \geq 0\}$. Further, the finite set of environmental states is $S = \{1, 2, \ldots, l\}$, where l is the dimension of the state-space. Each state has a corresponding linear wear rate. Thus $\mathscr{Z}$ forms a finite, continuous-time Markov chain that is assumed to fully characterize the ambient environment. The transitions are governed by an infinitesimal generator matrix $\mathbf{Q}$ with an initial distribution $\boldsymbol{\alpha}$. The transition probability functions for $\mathscr{Z}$ are denoted by

$$\pi_{i,j}(t) = \Pr\{Z_t = j \mid Z_0 = i\}, \quad i, j \in S.$$

For each state $j \in S$ there is a corresponding positive wear rate $r(j)$. Together, the wear rates form a vector $\boldsymbol{r}$ of dimension l. These wear rates are placed along the diagonal of a matrix of zeros to form the diagonal matrix $\mathbf{R}_D = diag\{\boldsymbol{r}\}$. The ratio of the damage threshold, x, to the minimum wear rate is denoted by Λ, and is considered the maximum component lifetime. This is because Λ represents the time to failure if the component were subject to only the most benign damaging environment until failure and endured no damage from shocks. Λ is formally defined by

$$\Lambda = \frac{x}{\min\{r(i) : i \in S\}}. \tag{3.1}$$

The total accumulated wear W_t at time t is defined by

$$W_t = \int_0^t r(\mathcal{Z}_u)\, du.$$

Shocks arrive according to a Poisson process, $\{N_t : t \geq 0\}$, with rate λ. The sequence of independent and identically distributed (i.i.d.) random variables, $\{Y_n : n = 1, 2, \dots\}$, represents the individual random shock magnitude each with cumulative distribution function (c.d.f.) F_Y. The Laplace-Stieltjes transform $\widetilde{F}_Y(u)$ of $F_Y(y)$ is given by,

$$\tilde{F}_Y(u) = \int_0^\infty e^{-uy}\, dF_Y(y),$$

and the diagonal matrix $\tilde{\mathbf{F}}_D(u)$ is defined to be

$$\tilde{\mathbf{F}}_D(u) = diag\{\tilde{F}_Y(u)\}.$$

The total accumulated damage β_t at time t is

$$\beta_t = \sum_{n=0}^{N_t} Y_n,$$

and the total degradation X_t at time t, can then be expressed as

$$X_t = W_t + \beta_t.$$

The time required for X_t to first reach level x is denoted by the nonnegative random variable T_x; i.e.,

$$T_x \equiv \inf\{t : X_t \geq x\}.$$

Since no partial repairs are possible, the degradation process $\{X_t : t \geq 0\}$ monotonically increases until the first inspection after T_x or the start of the next replacement interval. Furthermore, the bivariate stochastic process $\{(X(t), Z(t)) : t \geq 0\}$ fully characterizes the state of the system at time t. Since X_t is monotonically increasing and all wear rates are positive, the event in which total cumulative damage is below the threshold is equivalent to the event in which total time elapsed is less than the time of failure, or

$$\{X_t \leq x\} \iff \{T_x \geq t\}. \tag{3.2}$$

The process describing the status of the component currently in use is the stochastic process $\{\psi(t) : t \geq 0\}$ given by

$$\psi(t) = \begin{cases} 1 & \text{if } X_t < x \\ 0 & \text{if } X_t \geq x \end{cases}.$$

Thus, $\psi(t)$ describes the state of the system at time t and is recognized as a right continuous up/down stochastic process. The nth component lifetime is denoted by L_n with a corresponding time of failure designated by F_n. Since the state of the system is only discerned at each inspection, the nth replacement epoch R_n occurs at the first inspection after the nth failure time, or

$$R_n = \min\{k : k\tau > F_n\}, \quad n = 0, 1, 2, \ldots.$$

with R_0 defined to be 0.

The entire process is graphically depicted through the sample paths displayed in Figure 3.1. Without loss of generality, assume that the environment considered in Figure 3.1 can be adequately described by three states. In this example, the first environment has the most corrosive associated linear wear rate, while the third environmental state has the most benign rate of wear. Besides demonstrating the

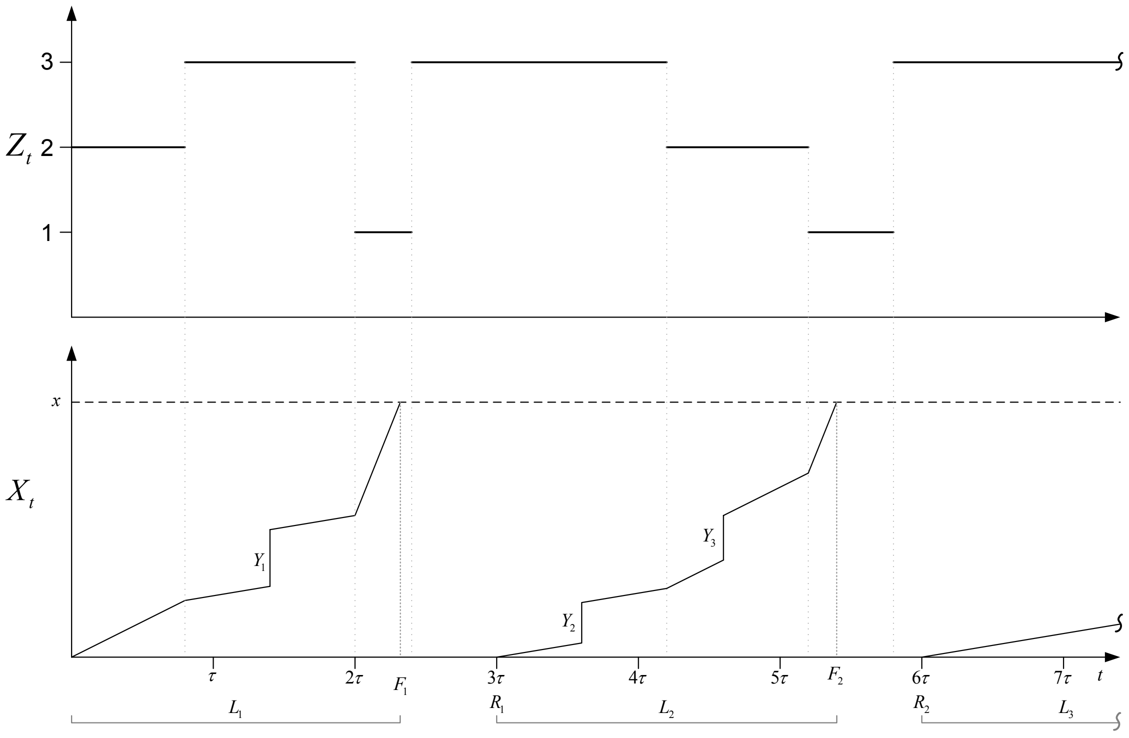

Figure 3.1 Sample paths of X_t and Z_t.

cumulative deterioration process as a combination of linear wear and random shocks subject to a control limit policy, the sample path analysis demonstrates that successive lifetimes do not form an i.i.d. sequence of random variables unless they begin their lifetimes in identical initial environmental states. Moreover, the dependency of failure and replacement epochs on component lifetimes is clear.

Also of interest is the embedded process representing the state of the environment at the nth replacement epoch, $\{\xi_n : n \geq 0\}$. Since there are a finite number of states, ξ_n forms an irreducible embedded discrete time Markov chain (DTMC) on S with transition probability matrix $\boldsymbol{P}$ and stationary distribution $\boldsymbol{p}$. An important result reported in [67] is that $\{(\xi_n, R_n) : n \geq 0\}$ forms a Markov renewal process. This is formally stated in Theorem 3.1 and a formal proof is available in [67]. This theorem is necessary in order to use an availability equation that requires a renewal process since the sequences $\{F_n : n \geq 0\}$ and $\{R_n : n \geq 0\}$ do not form a set of i.i.d.

component lifetimes unless the replacement epoch occurs when the environment is in the same initial state.

Theorem 3.1 *The bivariate process $\{(\xi_n, R_n) : n \geq 0\}$ is a Markov renewal process, and the continuous-time process $\{\psi(t) : t \geq 0\}$ is Markov-regenerative w.r.t $\{(\xi_n, R_n) : n \geq 0\}$*

Let $\mathbb{E}[\cdot]$ denote the expectation operator, and $\mathbb{E}_i[\cdot]$ denote the conditional expectation $\mathbb{E}[\cdot \mid Z_0 = i]$. The limiting average availability is defined by Leemis in [76] as,

$$\bar{A} = \lim_{t\to\infty} t^{-1} \int_0^t \mathbb{E}[\psi(w)]\, dw. \tag{3.3}$$

Theorem 3.2, which was proved by Kiessler et al. [68:704], expresses the limiting average availability as a direct consequence of the renewal reward theorem. Theorems 3.1 and 3.2 allow for the limiting average availability to be represented as the expected time to the first failure divided by the expected time to the first replacement, only if these expected values are conditioned on the initial state of the environment.

Theorem 3.2 *The limiting average availability is given by*

$$\bar{A}(\tau) = \lim_{t\to\infty} t^{-1} \int_0^t \mathbb{E}[\psi(w)]\, dw = \frac{\sum_{i=1}^{l} p_i\, \mathbb{E}_i[F_1]}{\sum_{i=1}^{l} p_i\, \mathbb{E}_i[R_1]}. \tag{3.4}$$

where $p_i = \lim_{n\to\infty} \Pr\{\xi_n = i\}$ with $i = 1, 2, \ldots, l$.

In order to compute equation (3.4) Kharoufeh et al. [67] derived explicit representations for $\mathbb{E}_i[F_1]$ and $\mathbb{E}_i[R_1]$ in terms of $\mathbf{Q}$, λ, $\tilde{\mathbf{F}}_Y(u)$, $\mathbf{R}_D$, x and τ.

The distribution function of the total random lifetime of the system is defined as

$$G(x,t) \equiv \Pr\{T_x \leq t\}$$

and the component lifetime distribution conditioned on the initial state i, is denoted as $G_i(\cdot) \equiv G(\cdot \mid Z_0 = i)$. The conditional distribution function can be written as,

$$G_i(x,t) \equiv \Pr\{T_x \le t \mid Z_0 = i\}, \quad i \in S.$$

Kharoufeh et al. [67] show that the failure time distribution satisfies a system of linear first-order partial differential equations. They solve this system using transform methods, yielding the Laplace-Stieltjes transform of the lifetime distribution. For further details, the reader is referred to [67]. This distribution can be used to determine the Laplace-Stieltjes transform of the conditional and unconditional mean lifetime with respect to x and also determine the conditional expectation of the first replacement epoch.

Component conditional and unconditional lifetime distributions are stated in Theorem 3.3. In this theorem, $\boldsymbol{\alpha}$ denotes the initial state probability vector, $\boldsymbol{e}$ denotes a column vector of ones, and $\boldsymbol{e}_i$ denotes the ith column in an identity matrix of dimension l.

Theorem 3.3 *The Laplace-Stieltjes transform of the unconditional and conditional first lifetime distributions, $G(x,t)$ and $G_i(x,t)$, with respect to x are, respectively,*

$$\tilde{G}(u,t) = 1 - \boldsymbol{\alpha} \exp\Big(\Big(\mathbf{Q} + \lambda(\tilde{\mathbf{F}}_D(u) - \mathbf{I}) - u\,\mathbf{R}_D\Big)\,t\Big)\,\boldsymbol{e} \tag{3.5}$$

and

$$\tilde{G}_i(u,t) = 1 - \boldsymbol{e}_i^T \exp\Big(\Big(\mathbf{Q} + \lambda(\tilde{\mathbf{F}}_D(u) - \mathbf{I}) - u\,\mathbf{R}_D\Big)\,t\Big)\,\boldsymbol{e}. \tag{3.6}$$

Theorem 3.1 requires that the expected times to first failure and first replacement must be conditioned on the initial environmental state, necessitating an expression for the transition probability matrix $\boldsymbol{P}$ for the DTMC $\{\xi_n : n \ge 0\}$. The

(i,k)th element of $\boldsymbol{P}$ is

$$p_{i,k} = \sum_{n=1}^{\gamma} \pi_{i,k}(n\tau) \Pr\left\{R_1 = n\tau \mid Z_0 = i\right\} \tag{3.7}$$

where, γ is the number of inspection intervals required to exceed the maximum component lifetime, or

$$\gamma \equiv \min\{n \geq 1 : n\tau \geq \Lambda\}. \tag{3.8}$$

Since γ determines the number of terms which must be summed according to equation (3.7), its size is very important to the overall computation speed. For use in a computational setting, the ceiling function can equivalently be used to define γ as

$$\gamma = \lceil \Lambda/\tau \rceil . \tag{3.9}$$

The probability $\Pr\left\{R_1 = n\tau \mid Z_0 = i\right\}$ is given by

$$\Pr\left\{R_1 = n\tau \mid Z_0 = i\right\} = G_i(x, n\tau) - G_i(x, (n-1)\tau) \equiv \Delta_i(x, n\tau). \tag{3.10}$$

Due to the structure of the process, the expected time of the first failure is

$$\mathbb{E}_i\left[F_1\right] = \mathbb{E}\left[T_x \mid Z_0 = i\right]. \tag{3.11}$$

By applying the definition of expectation and employing the properties of Laplace transforms to (3.6), the Laplace-Stieltjes transform of (3.11) becomes

$$\tilde{\mathbb{E}}\left[T_u\right] = \boldsymbol{\alpha}\left(u\mathbf{R}_D - \mathbf{Q} - \lambda\left(\tilde{\mathbf{F}}_D(u) - \mathbf{I}\right)\right)^{-1}\boldsymbol{e} \tag{3.12}$$

with the conditional representation,

$$\tilde{\mathbb{E}}_i\left[T_u\right] = \boldsymbol{e}_i^T\left(u\mathbf{R}_D - \mathbf{Q} - \lambda\left(\tilde{\mathbf{F}}_D(u) - \mathbf{I}\right)\right)^{-1}\boldsymbol{e}, \tag{3.13}$$

as shown in [67].

The sole expression remaining to determine the limiting average availability is the conditional expected time for the first replacement to occur. By proving that T_x is bounded above by x/r_1, where $r_1 = \min\{r(i) : i \in S\}$ is the minimum linear wear rate, and by conditioning, Kharoufeh et al. [67] compute $\mathbb{E}_i\left[R_1\right]$ as,

$$\mathbb{E}_i\left[R_1\right] = \tau\left(\gamma - \sum_{n=0}^{\gamma-1} G_i(x, n\tau)\right). \tag{3.14}$$

With expressions for the expected time of the first failure, and the expected time for the first replacement, the limiting average availability can now be computed. From Theorem 3.2 and the various components defined above, the limiting average availability can be expressed as

$$\begin{aligned}\bar{A}(\tau) &= \frac{\sum_{i=1}^{l} p_i\,\mathbb{E}_i\left[F_1\right]}{\sum_{i=1}^{l} p_i\,\mathbb{E}_i\left[R_1\right]} \\ &= \frac{\sum_{i=1}^{l} p_i\left(\mathcal{L}^{-1}\left\{u^{-1}\boldsymbol{e}_i^T\left(u\mathbf{R}_D - \mathbf{Q} - \lambda\left(\tilde{\mathbf{F}}_D(u) - \mathbf{I}\right)\right)^{-1}\boldsymbol{e}\right\}\right)}{\sum_{i=1}^{l} p_i\,\tau\left(\gamma - \sum_{n=0}^{\gamma-1}\mathcal{L}^{-1}\left\{u^{-1}\tilde{G}_i(u, n\tau)\right\}\right)}\end{aligned} \tag{3.15}$$

where $\mathcal{L}^{-1}\left\{\cdot\right\}$ is the inverse Laplace transform operator.

3.2 Cost Function Development

The objective of this research is to maximize the limiting average availability as represented in equation (3.15) by selection of the inter-inspection interval (τ) while remaining within an arbitrary budget constraint $\mathcal{B}$ and incorporating a realistic cost scheme. Further, for this computation to be operationally useful, it is necessary to compute the maximum limiting average availability with minimal computation time. This optimization will be accomplished by defining a formal nonlinear program using equation (3.15) as the objective function, constrained by cost and boundaries imposed by problem structure. Computational performance improvements are discussed in chapters 4 and 5.

In order to maximize availability for a particular inspection interval, it is necessary to consider a set of costs relevant to industrial or military applications that remain within an associated budget, $\mathcal{B}$. The renewal reward theorem [111] allows for consideration of the long-run costs. Three long-run costs per each cycle are considered: (1) fixed cost of replacing a failed component C_R, (2) cost of downtime C_D and (3) per-inspection cost C_I. The expected cycle time is equivalent to the expected time to the first replacement or,

$$\sum_{i=0}^{l} p_i \, \mathbb{E}_i \left[R_1 \right].$$

When divided by the expected cycle time, the sum of these three long-run costs forms the long-run expected cost per cycle, which is limited by the availability improvement budget constraint, $\mathcal{B}$. The long-run cost per cycle is,

$$\frac{C_R + C_I + C_D}{\mathbb{E}\left[R_1 \right]}.$$

The replacement cost, C_R, is understood as the cost to replace a failed component. Since a replacement marks the end of a cycle, this cost occurs exactly once per cycle and is therefore a fixed quantity, not dependent on τ. In most industrial settings this cost is much higher than the per-inspection cost and often higher than the cost per unit of downtime. Furthermore, this cost represents an aggregate of the labor and material costs involved in replacing a component.

A cost vital to industry and defense is the cost incurred per unit time for component to be in a non-operational state. In an industrial setting, this cost per unit of downtime could be the revenue lost from a machine producing needed inventory. In a military setting, this downtime could be the operational cost of not having the use of a mission-critical protection mechanism. The cost per unit of downtime, c_D varies widely depending on the criticality of the component. The total long-run cost of downtime C_D in a particular cycle is the cost per unit of downtime multiplied by the expected elapsed time between a system failure and the start of the next cycle. Thus, C_D is given,

$$C_D = c_D \, \mathbb{E}\left[R_1 - F_1\right] = c_D \, \mathbb{E}\left[R_1\right] - c_D \, \mathbb{E}\left[F_1\right].$$

Therefore, the greater the expected elapsed time between a hidden failure and the next inspection, the greater the long-run cost of downtime. A smaller τ presumably ensures a smaller downtime cost as more frequent inspections decrease the expected time from a failure to the next inspection.

The final long-run cost considered is the long-run expected cost per inspection C_I, which occurs only when the cycle time is large enough, or if $R_1 > \tau$. The long-run number of inspections in a cycle is the expected length of the cycle divided by the deterministic inter-inspection interval length. Since partial inspections have no meaning, the remainder of this quotient must be excluded. If a single inspection costs c_I, then the expected total cost due to inspections in a given cycle is c_I multiplied by the expected number of inspections within a given cycle. Mathematically, the inspection cost can be expressed using the floor function $\lfloor \cdot \rfloor$ as

$$C_I = c_I \left\lfloor \frac{\mathbb{E}\left[R_1\right]}{\tau} \right\rfloor.$$

The longer the periodic inspection interval, the lower the long-run cost due to inspections. Conversely, frequent inspections cause high inspection costs. In this manner, the per-inspection cost penalizes frequent inspections. In an industrial setting, a single per-inspection cost is often far lower than the cost of down-time and the cost of replacement.

Since successive unit lifetimes form an i.i.d. sequence of random variables only if they are placed into service in identical environmental states, the conditional expectations for the first failure and first replacement must be multiplied by the stationary probabilities p_i and summed to produce the unconditional expected times to first failure and first replacement. By the renewal reward theorem, the long-run cost is the sum each of the expected costs per cycle divided by the expected cycle length, which is,

$$\frac{\sum_{i=1}^{l} p_i \left(C_R + c_D\, \mathbb{E}_i\left[R_1\right] - c_D\, \mathbb{E}_i\left[F_1\right] + c_I \left\lfloor \frac{\mathbb{E}_i[R_1]}{\tau} \right\rfloor \right)}{\sum_{i=1}^{l} p_i\, \mathbb{E}_i\left[R_1\right]}. \tag{3.16}$$

It is important to note that this cost function is not convex, since the inspection cost is not a continuous function with respect to τ. Moreover, since equations for $\mathbb{E}_i\left[F_1\right]$ and $\mathbb{E}_i\left[R_1\right]$ are represented only in the form of Laplace transforms, equation (3.16) is not guaranteed to be smooth or continuous. As will be discussed in chapter 4, this precludes the application of traditional nonlinear optimization techniques.

3.3 Mathematical Programming Formulation

In order to maximize the limiting average availability as represented in equation (3.15) by selection of the inter-inspection interval, the following nonlinear program was constructed. The objective function requires x, $\mathbf{R}_D$, $\tilde{\mathbf{F}}_D(u)$, λ and $\mathbf{Q}$ to be defined *a priori* and evaluates the limiting average availability as a function of the decision variable, τ, the deterministic inspection interval. The full objective function is,

$$\bar{A}(\tau) = \frac{\sum_{i=1}^{l} p_i \left(\mathcal{L}^{-1} \left\{ u^{-1} \boldsymbol{e}_i^T \left(u\mathbf{R}_D - \mathbf{Q} - \lambda \left(\tilde{\mathbf{F}}_D(u) - \mathbf{I} \right) \right)^{-1} \boldsymbol{e} \right\} \right)}{\sum_{i=1}^{l} p_i\, \tau \left(\gamma - \sum_{n=0}^{\gamma-1} \mathcal{L}^{-1} \left\{ u^{-1} \tilde{G}_i(u, n\tau) \right\} \right)}$$

which can be written more compactly as

$$\bar{A}(\tau) = \frac{\sum_{i=1}^{l} p_i \, \mathbb{E}_i \left[F_1\right]}{\sum_{i=1}^{l} p_i \, \mathbb{E}_i \left[R_1\right]}.$$

3.3.1 Constraints

In order to form a complete optimization problem, bounds on τ were computed for the limiting average availability as presented by the stochastic degradation model. It was proved by [55] that T_x is bounded by Λ. Moreover, when τ reaches Λ, the value of γ is unity, and the expected time for the first replacement epoch, as presented in equation (3.14), is also Λ. As the value of τ increases past Λ by some positive valued quantity δ, the value of γ remains at one since one inspection interval is large enough to cover Λ or,

$$\gamma = \left\lceil \frac{\Lambda}{\Lambda + \delta} \right\rceil = 1, \quad \forall \quad \delta \geq 0.$$

Therefore, for some $\tau_1 \geq \Lambda$ where $\tau_1 = \Lambda + \delta$, equation (3.14) becomes

$$\mathbb{E}_i \left[R_1\right] = \tau_1 \left(\gamma - \sum_{n=0}^{\gamma-1} G_i(x, n\tau) \right) = \Lambda + \delta. \tag{3.17}$$

Therefore, the denominator of $\bar{A}(\tau)$ increases proportionally to δ and $\bar{A}(\tau)$ approaches zero since the numerator is constant with respect to τ. However, inspecting the system after a failure has occurred with probability one, does not make sense in an operational context. Thus, the upper bound of τ is set to Λ and $\mathbb{E}_i \left[R_1\right]$ is consequently less than or equal to Λ.

In order to maximize the limiting average availability, it is necessary to understand availability measures as τ approaches zero. In words, inspecting infinitely often generates an availability of one. This is understood in that, as τ gets smaller, the expected time until the next replacement approaches the expected time to the next failure and failures cease to be hidden in that they are instantly detected. However, while τ can get very small, it will always be positive with a finite $\mathcal{B}$. This is due to the inversely proportional relationship between C_I and τ. Inspection cost approaches infinity as τ gets small. That the decision variable remains finite over the interval of interest is an important property for several optimization methodologies.

Even though both the objective function and the cost constraint must be treated as black-box functions, both satisfy the Lipschitz condition, which is necessary to prove convergence for a number of derivative-free optimization methods. A function $f : I \to \mathbb{R}$ is said to be Lipschitz continuous if there exists a constant $K > 0$ such that, $|f(x) - f(y)| \leq K|x - y|$ for all x, y in the interval I. By definition, $\bar{A}(\tau)$ produces values in the region [0,1]. As established, the interval of $\mathbb{R}$ for this problem is restricted to $(0, \Lambda]$. Thus for any τ_1 and τ_2 in $(0, \Lambda]$,

$$\left|\bar{A}(\tau_1) - \bar{A}(\tau_2)\right| \leq 1$$

and the denominator is bounded by,

$$|\tau_1 - \tau_2| < \Lambda.$$

Therefore, with the Lipschitz constant $0 < K < \infty$,

$$\frac{\left|\bar{A}(\tau_1) - \bar{A}(\tau_1)\right|}{|\tau_1 - \tau_2|} < \frac{1}{\Lambda} < K \quad \tau_1, \tau_2 \in (0, \Lambda],$$

the objective function is Lipschitz continuous.

By the constraints imposed on τ, $\mathbb{E}_i\left[R_1\right] \leq \Lambda$. For a finite τ, $R_n > F_n$ and the expected cycle time will be positive as long as τ is nonzero. Therefore, $\mathbb{E}_i\left[R_1\right] - \mathbb{E}_i\left[F_1\right]$ will always be some finite number. Moreover, the finite budget constraint ensures the long-run cost is always finite. Thus, the cost constraint is also Lipschitz continuous since

$$\frac{|C(\tau_1) - C(\tau_2)|}{|\tau_1 - \tau_2|} < K \quad \tau_1, \tau_2 \in (0, \Lambda].$$

3.3.2 Nonlinear Programming Formulation

With the constraints and the decision variable defined, the full nonlinear program can be stated as

$$\text{Maximize} \quad \bar{A}(\tau) = \frac{\sum_{i=1}^{l} p_i \, \mathbb{E}_i\left[F_1\right]}{\sum_{i=1}^{l} p_i \, \mathbb{E}_i\left[R_1\right]} \tag{3.18}$$

subject to

$$\frac{\sum_{i=1}^{l} p_i \left(C_R + C_D(\mathbb{E}_i\left[R_1\right] - \mathbb{E}_i\left[F_1\right]) + C_I \lfloor \frac{\mathbb{E}_i[R_1]}{\tau} \rfloor \right)}{\sum_{i=1}^{l} p_i \, \mathbb{E}_i\left[R_1\right]} \leq \mathcal{B}$$

$$0 < \tau \leq \Lambda$$

This chapter has summarized the results of the stochastic degradation model presented by Kharoufeh et al. [67], in which degradation was the result of random shocks and environment-dependent linear wear. From this paper, the availability and reliability measures were presented and discussed. Then, a cost model to consider the long-run costs using the renewal reward theorem was developed. Constraints on τ were discussed and developed. The availability and reliability measures from the stochastic degradation model and the the cost constraint function were combined to formulate a nonlinear program.

However, the equations for $\mathbb{E}_i\left[F_1\right]$ and $\mathbb{E}_i\left[R_1\right]$ can be only represented using Laplace transforms and must consequently be numerically inverted to compute. Moreover, the transition probability matrix $\boldsymbol{P}$ requires computing the component lifetime distribution which also requires the numerical Laplace transform inversion. This presents considerable challenges and precludes the use of many standard optimization techniques. The next chapter will discuss a solution methodology and optimization strategies to solve the nonlinear program of section 3.3.

4. Solution Procedures

This chapter discusses the approach used to approximately solve the nonlinear programming problem presented in section 3.3. First, standard solution procedures for typical optimization problems are discussed. While modifications to these procedures for nonsmooth objective functions are promising, they are generally not well-suited for an objective function that exists only in the form of a Laplace transform. However, solutions may be obtained by using generalized pattern search, an approach that requires neither smooth functions nor derivative information and has proven convergence properties. After a discussion of pattern search, an algorithm directly implementing the analytical model of chapter 3 to compute the limiting average availability $\bar{A}$ is presented, followed by several strategies to improve computational performance. These result in an improved iterative algorithm for maximizing $\bar{A}(\tau)$ with a truncation method that dramatically decreases the computational requirements necessary to solve the original optimization problem. Finally, in order to better improve computational performance, two of the most often computed and consequently most computationally expensive algorithms are discussed: the matrix exponential and the Laplace transform inversion algorithm.

The optimization problem of Section 3.3 seeks to compute the maximum of the limiting average availability $\bar{A}(\tau)$ on the interval $(0, \Lambda]$ by selection of the value of τ that maximizes $\bar{A}(\tau)$, denoted by $\tau^\star$. Since the objective and cost functions are not necessarily smooth or even continuous, traditional necessary and sufficient conditions for optimality may not apply. Since the problem is nonconvex and treated as a *black box*, convergence to a global optimizer cannot be guaranteed [127]. Therefore, the variable $\tau^\star$ is assumed be only a local maximizer, which means that, for some $\varepsilon > 0$, $\bar{A}(\tau^\star) \geq \bar{A}(\tau)$ for all $\tau \in (0, \Lambda]$ satisfying $\|\tau - \tau^\star\| < \varepsilon$.

As discussed in chapter 3, the objective function in equation (3.18) involves a Laplace transform and contains a generally defined c.d.f. This adds considerable complexity to the overall optimization problem, since numerical methods must be used to compute inverse Laplace transforms and derivative information is not available. Moreover, the objective function may not be smooth, since no such assumption is made on the c.d.f.

Clearly, an analytical solution to (3.18) is preferred, in which elementary calculus yields a simple expression for the optimal τ. It is sometimes possible to apply partial fraction decomposition to a Laplace transform-based function in order to facilitate analytic inversion using Laplace transform tables. However, in this research, an analytical representation for neither the first replacement epoch ($\mathbb{E}_i\left[R_1\right]$) nor the first failure epoch ($\mathbb{E}_i\left[F_1\right]$) was attainable; consequently, numerical methods were required to compute both the objective function and the nonlinear cost constraint.

Many numerical algorithms exist for solving unconstrained optimization problems with differentiable functions, including gradient descent, Newton's method, quasi-Newton methods, and conjugate gradient methods. These techniques require an initial feasible point and use derivative information to generate a sequence of iterates that converge to a point satisfying certain optimality conditions. Extensions to these methods for constrained problems also appear throughout the literature (see [32], for example). If derivatives are not available, a common practice is to approximate them with finite differences. However, their use still assumes that derivatives exist, even though they are not available. Modifications of derivative-based methods for nonsmooth functions also exist [106, 103, 64, 63], but they generally require an explicit objective function to exist at every point on the interval of concern in the real domain.

Heuristics, such as local search, greedy algorithms, simulated annealing, ant colony optimization, tabu search and genetic algorithms (e.g., see [90]), are useful for problems in which other methods cannot be applied. They treat the objective function as a black box and are useful in practice for improving the objective function value; however, useful convergence properties are extremely rare [91], and the random nature of some of these methods would undoubtedly generate trial points τ close to zero, incurring unnecessary computational expense due to the associated large values of γ.

Among the multitude of direct search methods, three other approaches were considered: Nelder-Mead, golden section search, and generalized pattern search (GPS). Nelder-Mead is a flexible numerical method for optimizing multidimensional unconstrained problems and belongs to the more general class of direct search algorithms [98]. Further, Nelder-Mead is a commonly used for nonlinear optimization problems and known to perform well when the objective function is difficult or expensive to compute, not smooth, or when function evaluation is noisy [101]. Nelder-Mead can work effectively in these situations because it only uses function evaluations to reach a solution. Nelder-Mead has been shown to converge to the optimal solution when applied to strictly convex functions in one and two dimensions [74], but not in the general case. In fact, counter-examples exist [88] that show that Nelder-Mead cannot even guarantee convergence to a first-order stationary point.

Golden section search is a method specifically for locating the minimum point of a continuous function on a fixed interval. It requires no information about the derivative of the function. Golden section search starts with a function evaluation $f(x)$ at some point x in the larger of the two intervals (a, b) or (b, c). If $f(x) < f(b)$, then x replaces the midpoint b, and b becomes an end point. If $f(b) < f(x)$, then b remains the midpoint with x replacing one of the end points. Regardless of the outcome, the width of the bracketing interval will decrease. The procedure is repeated until the final width achieves a desired tolerance. If the new test point x is chosen to be the specific proportion of $(3 - \sqrt{5})/2$ (known as the *golden section*) along the larger sub-interval, measured from the mid-point b, then function values can be re-used, and the number of function values to achieve a desired level of accuracy is minimized. However, the method converges slowly, cannot handle nonlinear constraints, and is not extendable to multiple dimensions. For further details, the reader is referred to [105].

GPS is a derivative-free method originally introduced by Torczon [130] for unconstrained problems, who proved convergence of a subsequence of iterates to first-order stationary point. It has known convergence properties for a variety of problem classes, even when the objective function is nonsmooth [22]. Generalized pattern search methods iteratively search a set of points around the current iterative point for one that improves the objective function value. Because pattern search methods do not require derivative information to search this set of points, they are commonly used when evaluation of the objective function is expensive or when accurate approximation of derivatives is problematic. Consequently, pattern search can be applied to the problem in (3.18), even though the objective function requires inversions of several Laplace transforms.

Due to the shortcomings of the other methods described in this section, GPS was chosen as the optimization method for this thesis. The next section describes the algorithm in greater detail, and in a manner similar to that of [12] and [22]. Convergence results are discussed shortly thereafter.

4.1 Generalized Pattern Search

Consider the general nonlinear minimization problem,

$$\min_{x \in \Omega} f(x), \tag{4.1}$$

where $\Omega = \{x \in \mathbb{R}^n : \boldsymbol{\ell} \leq \mathbf{A}x \leq \boldsymbol{u}\}$, $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{A} \in \mathbb{Q}^{m \times n}$ is a rational matrix. Moreover, $\boldsymbol{\ell}$ and $\boldsymbol{u}$ are the lower and upper bounds of the constraints where $\boldsymbol{\ell}$ and $\boldsymbol{u}$ are $\in \{\mathbb{R}^m \cap \{\pm\infty\}\}$ and $\boldsymbol{\ell} \leq \boldsymbol{u}$.

GPS algorithms generate a sequence of iterates $\{x_k\}$ in $\mathbb{R}^n$ with nonincreasing objective function values. Each iteration is divided into an optional SEARCH step and a local POLL step. Both the SEARCH and POLL steps evaluate points on a mesh in order to find a mesh point with an improved function value. The *mesh* is constructed as a lattice of points in $\mathbb{R}^n$, based on a finite set of directions D that form a positive spanning set and a *mesh size parameter* $\Delta_k > 0$ that controls the fineness of the mesh. In this case, a positive spanning set refers to a set of vectors such that any vector in the space to be represented by a nonnegative linear combination of the vectors in the set.

By definition, nonnegative linear combinations of the elements of the set D span $\mathbb{R}^n$. The directions that form D can be arbitrarily chosen provided that each direction $d_j \in D$, $j = 1, 2, \ldots, |D|$, $d_j = \mathbf{G}\bar{z}_j$, where $\mathbf{G} \in \mathbb{R}^{n \times n}$ is a nonsingular matrix and $\bar{z}_j \in \mathbb{Z}^n$ is an integer vector. At iteration k, the mesh is centered around the current iterate $x_k \in \mathbb{R}^n$ and its fineness is parameterized through the mesh size parameter Δ_k. The mesh can then be represented as

$$M_k = \left\{ x_k + \Delta_k \mathbf{D}\, z : z \in \mathbb{Z}_+^{|D|} \right\} \tag{4.2}$$

where $\mathbb{Z}_+$ is the set of nonnegative integers. Note that in (4.2), the columns of the matrix $\mathbf{D}$ form the set D.

In the SEARCH step, GPS can evaluate any finite set of mesh points, and a number of strategies exist for generating trial points, including random search, genetic algorithms, Latin hypercube search, or orthogonal arrays. One popular strategy is to construct and optimize a *surrogate* function. A surrogate acts as an approximation to the objective function, but at a fraction of the cost. The use of surrogates often yields significant improvement in the objective function value early in the iteration process. For further details on surrogate functions, the reader is referred to [35].

If the SEARCH step fails to provide an improved mesh point, the POLL step is invoked. The POLL step is more rigidly defined and evaluates the neighboring mesh points for the current iterate. Use of positive spanning directions in the construction of these neighboring points provides the theoretical basis for the convergence of GPS. The POLL set at iteration k can be expressed as

$$\{x_k + \Delta_k d : d \in \mathbf{D}_k\}, \tag{4.3}$$

where $\mathbf{D}_k \subseteq D$ is also a matrix whose columns positively span $\mathbb{R}^n$. The POLL set is therefore composed of mesh points neighboring the current iterate x_k in the directions of the columns of $\mathbf{D}_k$, a multiple Δ_k away from the current iterate.

If the SEARCH and POLL step both fail, the incumbent solution is said to be a *mesh local optimizer* and the mesh is then refined by setting the mesh size parameter

$$\Delta_{k+1} = \theta^{w_k}\Delta_k, \tag{4.4}$$

where $\theta > 1$ is rational and $w_k \in \{w^-, w^- + 1, \ldots, -1\}$ for some w^-. An incumbent point x_k is replaced by x_{k+1} only if $f(x_{k+1}) < f(x_k)$, and x_{k+1} is termed an *improved mesh point*. If an improved mesh point is found in either step, then the mesh is either retained or coarsened by increasing the mesh size parameter according to equation (4.4) for some $w_k \in \{0, 1, \ldots, w^+\}$. It follows that for any $k \geq 0$ there exists an integer $r_k \in \mathbb{Z}$ such that $\Delta_k = \theta^{r_k}\Delta_0$. A simplified GPS algorithm for maximization is presented in Algorithm 1, in which $w_k = 0$. Algorithm 1 is presented as a maximization problem, which in general can simply be considered by minimizing the negative value of an objective function one is seeking to maximize.

The convergence analysis of pattern search is well-established in [21] and [130] and requires several assumptions. In order to discuss the convergence properties of GPS, several assumptions are necessary. First, all iterates produced by GPS must lie in a compact set [47]. This very common assumption holds as long as $\{x \in \Omega : f(x) \leq f(x_0)\}$ is compact. Second, if the matrix $\mathbf{G} = \mathbf{I}$ (as is usually the case), then the constraint matrix $\mathbf{A}$ must be rational. The final necessary assumption is that $f(x_0) < \infty$ for $x_0 \in \mathbb{R}^n$. Torczon [130] proved that, under the assumptions

```
Let: $D = \{d_1, d_2, \ldots, d_{|D|}\} \subset \mathbb{R}^n$ be a positive spanning set
Given: $x_0 \in \mathbb{R}^n$, $\Delta_0 > 0$, $\theta > 1$
for $k = 0, 1, \ldots$ do
  if $\exists\, d \in D$ such that $f(x_k + \Delta_k d) > f(x_k)$ then
    $x_{k+1} \leftarrow x_k + \Delta_k d$
    $\Delta_{k+1} \leftarrow \Delta_k$
  else
    $x_{k+1} \leftarrow x_k$
    $\Delta_{k+1} \leftarrow (1/\theta)\Delta_k$
  end if
end for
```

Figure 4.1 Algorithm 1: A basic GPS maximization algorithm for unconstrained problems.

discussed above, the mesh size parameter satisfies $\liminf_{k\to+\infty} \Delta_k = 0$. This leads to the main convergence result proved in [22] and restated as Theorem 4.1.

Theorem 4.1 *If $\hat{x}$ is any limit of a refining subsequence, and if d is any direction in D for which f at a* POLL *step was evaluated infinitely often in the subsequence, and if f is Lipschitz near $\hat{x}$, then the generalized directional derivative of f at $\hat{x}$ in the direction d is nonnegative.*

Theorem 4.1 is a directional result only, and is unsufficient for convergence to a stationary point. However, since the optimization problem considered in this thesis has only one decision variable, $D = \{-1, 1\}$ is the entire set of normalized positive spanning directions in $\mathbb{R}$ and Theorem 4.1 applies to both of them. Audet [20] proved convergence to a Clarke [46] first-order stationary point in the one-dimensional case for unconstrained problems. A similar argument based on [14] can be made to show convergence under mild conditions in the one-dimensional case to a local minimizer.

GPS was extended in [78] and [79] to problems with bound and linear constraints, respectively. To handle these constraints while maintaining convergence properties, infeasible points are discarded without being evaluated, and search directions are chosen so as to conform to the geometry of the nearby constraint boundaries. In one dimension, conforming directions are moot, since there are only two normalized directions. Extending GPS to general nonlinear constraints requires augmentation of the algorithm, such as in [80], [24], and [23]; however, in one direction, this becomes moot for the same reason as that for linear constraints. Thus, in one dimension, it is sufficient to use the basic GPS algorithm while ignoring infeasible points.

The NOMADm optimization software [11, 13], written in MATLAB® was used to implement the pattern search procedure used in this thesis. NOMADm is specifically designed to numerically solve nonlinear and mixed variable optimization problems via an implementation of the class of mesh adaptive direct search (MADS) algorithms. GPS is a subclass of MADS [25], in which POLL directions are restricted to a uniformly bounded finite set. In fact, in one dimension MADS and GPS essentially coincide.

The mathematical model presented in chapter 3 must be converted into a MATLAB function that produces $\bar{A}(\tau)$ for the NOMADm optimizer. This is accomplished by first directly implementing the equations of chapter 3 into a single MATLAB function. However, naively computing the equations in their necessary order proved too computationally expensive and resulted in excessive run times. This motivated the need to investigate a number of computational improvements which are discussed in section 4.3.

4.2 Numerical Computation of $\bar{A}(\tau)$

Recall from chapter 3 that computation of $\bar{A}(\tau)$ requires values for $\boldsymbol{p}$, $\mathbb{E}_i\left[F_1\right]$ and $\mathbb{E}_i\left[R_1\right]$. First, a direct analytical implementation is presented that simply computes the equations in the necessary order to first obtain $\boldsymbol{P}$, which leads to the vector of stationary probabilities $\boldsymbol{p}$. Subsequently, the conditional expected time to the first failure $\mathbb{E}_i\left[F_1\right]$ and first replacement $\mathbb{E}_i\left[R_1\right]$ are computed. Direct computation turns out to be extremely computationally expensive, especially when the dimension of $\mathbf{Q}$ or the value of Λ is large. Thus, several computational techniques were developed and incorporated in an attempt to improve the algorithm. This is followed by a more detailed analysis of two functions which account for the majority of the computational complexity: the matrix exponential and Laplace transform.

In order to compare the computational complexity of various algorithms analysis, techniques have been developed to understand complexity growth for very large input values. The most common notation used in complexity analysis is referred to as *big-oh* and is defined as follows [121]. Given any two functions f and g, then $f(n)$ is $\mathcal{O}(g(n))$ if there exist positive constants C and k such that:

$$f(n) \leq Cg(n) \quad \text{whenever } n > k, \text{ and } \{n, k\} \subseteq \mathbb{Z}.$$

While big-oh notation provides insight into the relative rates of growth between algorithms, it does not describe the actual characteristics for best and average cases for individual run times. First, many algorithms are simply too complex to analyze mathematically since they are often comprised of many sub-functions implemented in different languages. Second, it is difficult to qualitatively account for memory management and the interaction between hardware architectures and specific codes. For example, complexity analysis cannot factor in the effect on paging as virtual memory usage grows. Third, asymptotic complexity analysis does not give insight into the average run time. Fourth, asymptotic complexity analysis does not give any indication of the run time or efficiency of a particular algorithm, only an understanding of how complexity growth varies with increasing input. For example, one algorithm may be extremely fast and another cripplingly slow at a particular computation but have the same algorithmic order. Because of the disadvantages discussed above, only computational benchmarks comparing the actual run times of distinct algorithms, when combined with asymptotic complexity analysis, provide good insight into the relative performance between algorithms.

Directly computing the mathematical model presented in section 3.1 to solve the nonlinear program is extremely computationally expensive. The full algorithm is listed in Algorithm 2. It is comprised of five different stages, each with a particular computational result. The first stage populates the transition probability matrix $\boldsymbol{P}$ according to equation (3.7). The complexity of this stage is $\mathcal{O}(l^2\gamma\rho(l))$ if $\mathcal{O}(\rho(l))$ is the state-dependent computational burden for the particular matrix exponential algorithm needed to calculate $\pi_{i,k}(t)$. This stage is the most computationally complex in the algorithm. Within the third nested loop there are two Laplace inversion calls and one matrix exponential call on lines 6 and 7 of Algorithm 2, which account for the primary computational expense of the entire algorithm. For this reason, algorithms for these two function calls are examined carefully in section 4.4.

The complexity analysis above reveals sensitivity to the state-size when building the transition probability matrix $\boldsymbol{P}$. For each element of $\boldsymbol{P}$, equation (3.7) must be computed. The variable γ defined by equation (3.8) represents the number of inspections that can occur if the stochastically degrading component reaches its maximum possible lifetime. The maximum lifetime Λ is x/r_1. Therefore if Λ is much larger than τ, γ is also large. Since equation (3.7) involves a summation of γ terms, the overall computational expense is very sensitive to this quantity.

Stage 2 of the algorithm, described on lines 13-16, computes the stationary distribution $\boldsymbol{p}$ of $\boldsymbol{P}$. This is simply the solution to the equations,

$$\pi_j = \sum_{i\in S} \pi_i p_{ij} \qquad\qquad \sum_{i\in S} \pi_i = 1$$

presented in [72]. These steps are easily implemented and do not incur much computational cost.

Stages 3 and 4 compute equations (3.13) and (3.14) which produce $\mathbb{E}_i\left[F_1\right]$ and $\mathbb{E}_i\left[R_1\right]$, respectively. Equation (3.13) is implemented on lines 17 to 19 and equation (3.14) is implemented on lines 20-26. The primary driver for computational

```
Input: $\tau$
$\gamma \leftarrow \lceil x/(r_1\tau) \rceil$
for $i = 1$ to $l$ do
    for $k = 1$ to $l$ do
        for $j = 2$ to $\gamma$ do
            $\boldsymbol{\pi} \leftarrow \mathrm{expm}(\mathbf{Q} j\tau)$
            $X_j \leftarrow \left(\mathcal{L}^{-1}\left\{u^{-1}\tilde{G}_i(u, j\tau)\right\} - \mathcal{L}^{-1}\left\{u^{-1}\tilde{G}_i(u, (j-1)\tau)\right\}\right)\boldsymbol{\pi}_{i,k}$
        end for
        $\boldsymbol{\pi} \leftarrow \mathrm{expm}(\mathbf{Q}\tau)$
        $\mathbf{P}_{i,k} \leftarrow \sum X + \boldsymbol{\pi}_{i,k}\left(\mathcal{L}^{-1}\left\{u^{-1}\tilde{G}_i(u, \tau)\right\}\right)$
    end for
end for
$\hat{\mathbf{P}} \leftarrow \mathbf{P} - \mathbf{I}$
Set all elements in the last column of $\hat{\mathbf{P}}$ to one
Initialize $\boldsymbol{p}$ and set the last element to one
$\boldsymbol{p} \leftarrow \boldsymbol{p}\,\hat{\mathbf{P}}^{-1}$
for $i = 1$ to $l$ do
    Set the $i$th element of $\mathbb{E}[F_1]$ to $\mathcal{L}^{-1}\left\{u^{-1}\boldsymbol{e}_i\left(u\mathbf{R}_D - \mathbf{Q} - \lambda(\widetilde{\mathbf{F}}_D(u) - \mathbf{I})\right)^{-1}\boldsymbol{e}\right\}$
end for
for $i = 1$ to $l$ do
    for $j = 1$ to $\gamma - 1$ do
        $S_j \leftarrow \mathcal{L}^{-1}\left\{u^{-1}\tilde{G}_i(u, j\tau)\right\}$
    end for
    Set the $i$th element of $\mathbb{E}[R_1]$ to $\tau(\gamma - \sum S)$
end for
$A \leftarrow (\boldsymbol{p}\,\mathbb{E}[F_1])/(\boldsymbol{p}\,\mathbb{E}[R_1])$
```

Figure 4.2 Algorithm 2: Direct algorithm for evaluating $\bar{A}(\tau)$.

complexity in these two stages is the Laplace transform inversion algorithm which is examined in section 4.4. The complexity of stage 1 is much greater than either of these stages, even though the asymptotic complexity of the numerical computation to evaluate equation (3.14) is only of order $\mathcal{O}(l(\gamma-1))$. The final stage of Algorithm 2 is to bring together $\mathbb{E}_i[R_1]$, $\mathbb{E}_i[F_1]$ and $\boldsymbol{p}$ to compute the limiting average availability $\bar{A}$ for a particular τ. Equation (3.4) is computed on line 26 without much computational expense.

4.3 Computational Enhancements

Algorithm 2 was presented as the objective function for NOMADm and several numerical experiments were performed, each resulting in excessively long run times, even for small state sizes. This motivated the need for a more efficient implementation of the objective function. This section documents several strategies taken to reduce the computational complexity of Algorithm 2, since this should reduce overall run times for the optimization procedure.

The first strategy was to focus on implementing a series of general computational improvement techniques. MATLAB is an interpretive language with more computational overhead than an associated compiled language. Consequently, there are several techniques that take advantage of the way MATLAB executes code to vastly improve efficiency. These techniques include the use of functions, *vectorization*, and use of the most appropriate operator.

One early improvement made was to place all code into functions. MATLAB code executes more quickly when implemented in a function rather than a script. Every time a script is used, the entire script is loaded into memory and evaluated one line at a time with a large amount of overhead processing. Functions, on the other hand, are compiled into pseudo-code and loaded into memory once, with private variables accessible only to the current function. Therefore, the more often a procedure is called, the greater the advantage of using a function instead of a script.

As MATLAB is designed to operate on matrices, it is important to use vector operations whenever possible. Vectorization can increase computational speed by several orders of magnitude. For example, while `for` loops are intuitive, they take much longer to execute, since each line must be sent separately to the processor. If a vector operation is used instead, MATLAB processes this operation as a single command. Vectorization was most helpful in the Laplace transform inversion algorithm where several `for` loops were replaced by different vector commands.

An example of replacing an inefficient operator with one more suited to the computation at hand was to replace the `inv` command on line 16 of Algorithm 2 with the matrix right division operator. This exchanges the full `LAPACK` inversion algorithm for a custom MATLAB procedure that exploits the structure of the matrix to find the most efficient method of solving the linear system (such as Gaussian elimination with partial pivoting if the matrix is square).

The second strategy used to increase computational speed was to carefully analyze locations of heavy computational cost and more efficiently implement them. This included relocating computations from the more frequently called sections of Algorithm 2 to a pre-processing block that is only executed once per function call. NOMADm provides for the use of a parameter file that loads at the start of a run, and each value that requires computation only once was moved to this file. Since memory retrieval is much faster than computations, even seemingly small computations were moved into this file.

The MATLAB profiler measures the computational cost of each line of code. When applied to the MATLAB implementation of Algorithm 2, line 7 was discovered to be a key computational cost with two different inversion calls. Since the difference between the two calls was one iteration, this line was changed to evaluate one inversion call and to take the differences all in one step outside the third nested loop of stage one in Algorithm 2. This led to a subsequent analysis showing that, by adjusting the indices of the inner loop, one `expm` call could be removed from the second nested loop of this same stage. A similar analysis was conducted throughout all sub-functions.

While these strategies focused on making Algorithm 2 run more quickly, the third strategy focused on changing the algorithm itself to improve efficiency. From MATLAB profiler measurements, it was clear that most computational cost was coming, as expected, from the third nested loop of stage 1. Upon observation, the difference between each function call and the next quickly approached zero and decreased exponentially. This formed the central idea for the third strategy, to truncate this series at a given tolerance level ε, effectively turning Algorithm 2 from a direct method to an iterative one. The resulting *truncation method* provided savings proportional to increasing l and Λ. The truncation method can dramatically reduce the number of terms which must be summed by approximating the (i,k)th element of $\boldsymbol{P}$ according to

$$\begin{aligned} p_{i,k} &= \sum_{n=1}^{\gamma} \pi_{i,k}(n\tau)\mathbb{P}_i[R_1 = n\tau] \\ &\approx \sum_{n=1}^{N} \pi_{i,k}(n\tau)\left(G_i(x,n\tau) - G_i(x,(n-1)\tau)\right) \end{aligned}$$

where $N < \gamma$ is chosen so that $G_i(x,N\tau)-G_i(x,(N-1)\tau) \leq \varepsilon$. A sensitivity analysis was conducted and the value $\varepsilon = 10^{-4}$ was chosen in order to improve computational speed to the maximum degree possible, while not introducing noticeable error.

4.4 Matrix Exponential and Laplace Transforms

The final strategy for improving computational performance was to analyze the two most costly functions: the matrix exponential and the inverse Laplace transform.

```
Input: $\tau$
Precalculate: $\mathbb{E}[F_1]$ for all states
Set: $\gamma = \lceil x/(r_1\tau) \rceil$, $\boldsymbol{p}$
for $w = 1$ to $l$ do
    Clear $\boldsymbol{\alpha}$ and set the $n$th element of $\boldsymbol{\alpha}$ to 1
    for $k = 1$ to $l$ do
        for $n = 1$ to $\gamma$ do
            $\boldsymbol{\pi} \leftarrow$ Matrix Exponential of $\mathbf{Q}t$
            $X_n \leftarrow \mathcal{L}^{-1}\left\{u^{-1}\tilde{G}_i(u, n\tau)\right\}$
            $\mathcal{P}_n \leftarrow \pi_{w,k}$
            if $X_n - X_{n-1} \leq \epsilon$ then
                Break
            end if
        end for
        $p_{w,k} \leftarrow \sum_{r=2}^{\gamma}\left[(X_r - X_{r-1})\mathcal{P}_r\right] + \mathcal{P}_1 X_1$
    end for
end for
$\hat{\boldsymbol{P}} \leftarrow \boldsymbol{P} - \mathbf{I}$
Set each element of the last column of $\hat{\boldsymbol{P}}$ to one
$\boldsymbol{p} \leftarrow \boldsymbol{p}/\hat{\boldsymbol{P}}$
for $i = 1$ to $l$ do
    for $n = 1$ to $(\gamma - 1)$ do
        $S_n = \mathcal{L}^{-1}\left\{u^{-1}\tilde{G}_i(u, n\tau)\right\}$
    end for
    Set the $i$th element of $\mathbb{E}[R_1]$ to $\tau(\gamma - \sum S)$
end for
Availability $\leftarrow -(\boldsymbol{p}\,\mathbb{E}[F_1])/(\boldsymbol{p}\,\mathbb{E}[R_1])$
Cost $\leftarrow (\boldsymbol{p}\,(c_D(\mathbb{E}[R_1] - \mathbb{E}[F_1]) + c_I\lfloor \mathbb{E}[R_1]/\tau \rfloor + c_R\boldsymbol{e}))/(\boldsymbol{p}\,\mathbb{E}[R_1]) - \mathcal{B}$
```

Figure 4.3 Algorithm 3: Improved algorithm to compute $\bar{A}(\tau)$.

The following subsection reviews the associated numerical methods for each of these functions.

4.4.1 Computation of the Matrix Exponential

In order to numerically calculate the availability measures discussed in section 3.1, it is necessary to obtain the transition probability matrix for $\{\xi_n : n \geq 0\}$, which is computed according to equation (3.7). The matrix exponential is also a necessary operation to compute a component lifetime, per equation (3.3). The matrix exponential is defined by

$$e^{\mathbf{Q}\,t} \equiv \sum_{k=0}^{\infty} \frac{(\mathbf{Q}\,t)^k}{k!}. \tag{4.5}$$

In order to compute the values of $\boldsymbol{P}$ per equation (3.7), equation (4.5) must be numerically computed for each unique combination of states and for each unique value of τ from 1 up to γ. The limiting complexity of matrix exponential evaluations is $\mathcal{O}(l^2\gamma)$ or potentially varying cubically with the input. Therefore, for environments with even moderate state sizes (e.g., more than 5) and moderately-sized values of γ, the matrix exponential is a key driver of computational requirements. For this reason, a thorough analysis of the matrix exponential was conducted.

Numerous techniques for computing the matrix exponential have been presented in the literature [99, 133, 72, 128, 42, 92]. Moler and Van Loan [92] in particular surveyed nineteen different methods. A common method shared by many solutions uses matrix-scaling and powering to first form $e^{\mathbf{Q}t}$. Ordinary differential equation (ODE) solvers, can compute the matrix exponential directly without explicitly forming $e^{\mathbf{Q}t}$. According to [135], the stability of the matrix exponential is sensitive to perturbations in $\mathbf{Q}$, with the norm of $\mathbf{Q}$ being the most critical measure of stability. According to [135], if $\mathbf{Q}$ is a defective matrix (i.e., $\mathbf{Q}$ does not have a full set of eigenvectors), then the matrix exponential is relatively poorly conditioned.

Several methods were examined for computing the matrix exponential, each with a distinct set of properties. Specifically, these methods include a simple Taylor series expansion, the method of eigenvalues and eigenvectors, the uniformization method and the Padé approximation via repeated scaling and squaring. Computing the matrix exponential using these four different methods serves two primary purposes. First, since the matrix exponential is critical to the overall cost of computing $\bar{A}(\tau)$, it is important to use the most stable and efficient method available. Second, by using four different methods, it is possible to compare results and mitigate the risk of a particular method causing unacceptable error propagation or instability.

Computation of the matrix exponential via Taylor series is the most simple, but often slowest and least accurate method studied. According to the classic text [57], it is only useful for theoretical comparisons. Moler and Van Loan [92] demonstrate that this implementation is subject to *catastrophic cancellation* as truncation errors are greatly magnified as the factorial of the denominator increases. Despite these shortcomings, the simplicity of the Taylor series method facilitates error analysis and provides a lower-bound from which to understand other methods. A simple expression from which to calculate bounds on the Taylor series truncation error is given in [81]. Furthermore, the Taylor series method illustrates the classic definition for the matrix exponential and exposes pitfalls of computational round-off error.

The theory of the Taylor series method is straightforward. From elementary calculus, for any matrix $\mathbf{\Upsilon}$ and scalar t one may compute $e^{\mathbf{\Upsilon}t}$ by the sum of powers,

$$e^{\mathbf{\Upsilon}t} = \sum_{k=0}^{\infty} \frac{(\mathbf{\Upsilon}t)^k}{k!}. \tag{4.6}$$

The most simple way to deal with equation (4.6) is to add terms until there is no machine-detectable contribution by subsequent terms. The marginal contribution of each term is defined by the matrix $\mathbf{F}$ and the cumulative sum of the expansion is defined by the matrix $\mathbf{E}$. By establishing a non-strict boundary and using the matrix

```
$\mathbf{F} \leftarrow \mathbf{I}$
$k \leftarrow 1$
while $\|\mathbf{F}\|_1 > 0$ do
    $\mathbf{E} \leftarrow \mathbf{E} + \mathbf{F}$
    $\mathbf{F} \leftarrow \mathbf{QF}/k$
    $k \leftarrow k + 1$
end while
```

Figure 4.4 Algorithm 4: Matrix exponential via Taylor series.

1-norm (i.e., the largest column sum), the condition $\|\mathbf{F}\|_1 > 0$ stops computation when all components of **F** are zero to machine precision. The MathWorks corporation provides a simple MATLAB implementation of the Taylor series method which is summarized in Algorithm 4. Results of the Taylor series method for five different example problems are presented in chapter 5.

Matrix exponentiation via eigenvalues and eigenvectors is discussed in [72] and [128]. It is most stable with a small state size and when $\boldsymbol{\Upsilon} t$ has distinct eigenvalues [128]. Unlike the Taylor series and uniformization methods, the method of eigenvalues and eigenvectors can handle large values of t, as is often the case for reliability applications. However, a small matrix size is preferred, since the size of the matrix significantly impacts the cost of determining eigenvectors and eigenvalues.

The method of eigenvalues and eigenvectors is built on the relationship that the matrix exponential of a diagonal matrix is the diagonal matrix of element exponentials. Consequently, the matrix exponential is easily computed if the eigenvalues of $\mathbf{Q}$ are known. Assume $\mathbf{Q} \in \mathbb{R}^{n \times n}$ has n distinct eigenvalues ν_j, $j = 1, 2, \ldots, n$. Then by definition [89], if $\mathbf{Q}$ is not defective (i.e., it has a full set of eigenvectors), then

$$\mathbf{Q}_j s_j = \nu_j s_j, \quad j = 1, 2, \ldots, n,$$

or

$$\mathbf{Q} = \mathbf{S}\boldsymbol{\Gamma}\mathbf{S}^{-1}, \tag{4.7}$$

where $\mathbf{\Gamma} = \text{diag}\{\nu_1, \nu_2, \ldots, \nu_n\}$ and $\mathbf{S}$ is the full matrix of eigenvectors in where the jth column is the right eigenvector s_j corresponding to the eigenvalue ν_j, $j = 1, 2, \ldots, n$. Exploiting the properties of diagonal matrices, equation (4.7) can be written more generally [72] as,

$$\mathbf{Q}^k = \mathbf{S}\mathbf{\Gamma}^k\mathbf{S}^{-1}. \tag{4.8}$$

Accordingly, equation (4.5) can be written as,

$$\begin{aligned} e^{\mathbf{Q}t} &= \mathbf{S}\mathbf{S}^{-1} + \sum_{r=1}^{\infty} \mathbf{S}\frac{\mathbf{\Gamma}^r t^r}{r!}\mathbf{S}^{-1} \\ &= \mathbf{S}\left[\mathbf{I} + \sum_{r=1}^{\infty} \frac{\mathbf{\Gamma}^r t^r}{r!}\right]\mathbf{S}^{-1} \\ &= \mathbf{S}e^{\mathbf{\Gamma}t}\mathbf{S}^{-1}. \end{aligned} \tag{4.9}$$

Equation (4.9) is simple to calculate, since $\mathbf{\Gamma}^r = \text{diag}\{\nu_1^r, \nu_2^r, \ldots, \nu_n^r\}$, and since the matrix exponential of a diagonal matrix is the diagonal matrix of element exponentials, it follows that

$$e^{\mathbf{\Gamma}t} = \text{diag}\left\{e^{\nu_1 t}, e^{\nu_2 t}, \ldots, e^{\nu_n t}\right\}. \tag{4.10}$$

In MATLAB, the eigenvalues and eigenvectors can be easily computed using the built-in function `eig`, which uses a series of `LAPACK` routines to compute the eigenvalues and eigenvectors, depending on the properties of $\mathbf{Q}$. For details, see the `LAPACK` user's guide [16].

The main advantage of the method of eigenvalues and eigenvectors [134] is that once the eigenvalues and eigenvectors are determined, the probability distribution can be quickly calculated for any value of t in one matrix-vector multiplication. However, in this research, the transient solution is required only at the single terminal point. The primary weakness for this method is that $\mathbf{Q}$ must not be defective and according to [128], a small condition number cond($\mathbf{S}$) of $\mathbf{S}$ (where cond($\mathbf{S}$) $\equiv$

```
$[\mathbf{V},\mathbf{D}] \leftarrow \mathrm{eig}(\mathbf{Q})$
$\mathbf{E} \leftarrow \mathbf{V}\,\mathrm{diag}\,\{\exp(\mathrm{diag}\,\{\mathbf{D}\})\}\,/\mathbf{V}$
```

Figure 4.5 Algorithm 5: Matrix exponential via eigenvalues and eigenvectors.

$\|\mathbf{S}\|\,\|\mathbf{S}^{-1}\|$) is required to prevent small rounding errors from compounding. Moler and Van Loan [92] demonstrate the effect of large condition numbers of $\mathbf{S}$ with numerical examples. A MATLAB implementation of this method is summarized in Algorithm 5. Results from the implementation of this algorithm are presented in chapter 5.

The uniformization method (also called Jensen's method) is thoroughly discussed in the literature [34, 72, 99, 128, 73, 85]. One of the key strengths of uniformization is its simple implementation. It often outperforms other methods [128] and performs especially well when $\mathbf{Q}$ is large [73]. Furthermore, uniformization requires only modest memory allocation and provides an extremely stable and efficient method of computing the matrix exponential [72]. The primary disadvantage is that large values of νt cause considerable roundoff errors. According to Kulkarni [72], if $\nu t > 250$, $e^{-\nu t}$ is potentially less than the smallest computable floating point number which is often on the order of 10^{-300}.

The key requirement for uniformization is that the diagonal elements of $\mathbf{Q}$ be bounded; i.e., if η_{jj} is the jth diagonal element of $\mathbf{Q}$, $j = 1, 2, \ldots, n$, then

$$\|\eta_{jj}\| \le \nu < \infty \tag{4.11}$$

must hold for each state j and for some $\nu > 0$. If equation (4.11) holds, $\mathbf{Q}$ is said to be *uniformizable*, which is always the case if the state space is finite.

Kulkarni presents the following procedure in [72]. First, set ν to

$$\nu = \max_{1\le i\le l}\{-\eta_{ii}\}, \tag{4.12}$$

and define the discretized stochastic transition probability matrix $\boldsymbol{\Upsilon}$ as

$$\boldsymbol{\Upsilon} \equiv \mathbf{I} + \frac{1}{\nu}\mathbf{Q}, \tag{4.13}$$

which produces $\mathbf{Q} = \nu(\boldsymbol{\Upsilon} - \mathbf{I})$. Since, by the properties of exponentials,

$$e^{\mathbf{Q}t} = e^{t\nu\boldsymbol{\Upsilon} - t\nu\mathbf{I}} = e^{-t\nu}\, e^{(t\nu)\boldsymbol{\Upsilon}},$$

the *uniformization equation* is expressed as

$$\boldsymbol{\Upsilon}(t) = \exp\left\{\mathbf{Q}t\right\} = \sum_{k=0}^{\infty} \boldsymbol{\Upsilon}^k e^{-\nu t}\frac{(\nu t)^k}{k!} \tag{4.14}$$

where the infinite sum in equation (4.14) contains the Poisson probability mass function,

$$\Pr\left\{N(t) = k\right\} = \frac{(\nu t)^k}{k!}\exp\left(-\nu t\right).$$

A distinct advantage of uniformization is that it lends great control over truncation error. Stewart [128] presents theoretical bounds for the truncation error, yet due to machine round-off errors, the computed result may be much greater than these bounds. However, Stewart [128] states that roundoff error is generally not a problem since numerical operations involve no negative numbers without subtraction and $\|\boldsymbol{\Upsilon}\| \le 1$. In order to determine at which value M to truncate the sum, so that error does not exceed some value ε, the Poisson distribution can be used [72] in the equation,

$$\sum_{k=0}^{M} \frac{(\nu t)^k}{k!} e^{-\nu t} > 1 - \varepsilon. \tag{4.15}$$

The uniformization method, shown in Algorithms 6 and 7, was implemented in Matlab as presented in [99] and [128]. Algorithm 6 determines the value of M in lines 4 through 9, the value of $\boldsymbol{\Upsilon}$ according to equation (4.13) in line 10, and the value of ν according to (4.12) in line 3. Algorithm 6 is called only once per calculation of $\bar{A}(\tau)$. The uniformization equation (4.14) is implemented in Algorithm 7 and is called with every matrix exponential call. Further code was implemented on the recommendation of [99] to condition the output to ensure that $\boldsymbol{\Upsilon}$ retains properties of a stochastic matrix. Stewart [128] shows that the complexity of the uniformization method is $\mathcal{O}(n^2)$ and requires $M(l + n_z)$ multiplications, where n_z is the number of nonzero elements in $\boldsymbol{\Upsilon}$.

```
Input: $\tau$, $\mathbf{Q}$, $\varepsilon$
Set: $K = 0$, $\upsilon = 1$ and $\sigma = 1$
$\nu \leftarrow \max(-\operatorname{diag}\{\mathbf{Q}\})$
$\eta \leftarrow (1-\varepsilon)/(\exp(-\nu\tau))$
while $\sigma \leq \eta$ do
    $K \leftarrow K + 1$
    $\upsilon \leftarrow (\upsilon\nu\tau)/K$
    $\sigma \leftarrow \sigma + \upsilon$
end while
$\boldsymbol{\Upsilon} \leftarrow (\nu^{-1})(\nu\mathbf{I} + \mathbf{Q})$
```

Figure 4.6 Algorithm 6: Determine uniformization parameters.

```
Input: $\tau$, $\nu$ ,$\boldsymbol{\Upsilon}$ ,$n$
Set: $\boldsymbol{y} = \mathbf{I}$
for $k = 1$ to $n$ do
    $\boldsymbol{y} \leftarrow (\boldsymbol{y}\boldsymbol{\Upsilon}\nu\tau)/k$
    $\boldsymbol{\pi} \leftarrow \boldsymbol{\pi} + y$
end for
$\mathbf{E} \leftarrow \exp(-\nu\tau)\,\boldsymbol{\pi}$
```

Figure 4.7 Algorithm 7: Matrix exponential via Uniformization.

The final matrix exponentiation method is the Padé approximation via repeated scaling and squaring. This method is the default executed by the MATLAB `expm` command and is the one recommended by Moler and and Van Loan [92]. The Padé approximation, without a method to control roundoff error, suffers the same fate as the Taylor series method in that roundoff error worsens as $\|\mathbf{Q}\|\, t$ increases [27].

This error can be mitigated by exploiting a property of the exponential function,

$$\exp(\mathbf{Q}) = \exp(\mathbf{Q}/n)^n. \tag{4.16}$$

A value of n can be chosen to be a power of two for which $\exp(\mathbf{Q}/n)$ can be accurately computed [92]. From this, the matrix $\exp(\mathbf{Q}/n)^n$ can be formed by repeated squaring. Thus, this mechanism scales $\mathbf{Q}$ by a power of two to reduce the norm to order one, computes a Padé approximation to the matrix exponential, and then repeatedly squares to undo the effect of scaling.

A development of the Padé approximation follows. The (a, b) Padé approximation to the matrix exponential is, by definition [17], the unique rational function,

$$R_{a,b}(\mathbf{Q}t) \equiv \frac{N_{a,b}(\mathbf{Q}t)}{D_{a,b}(\mathbf{Q}t)}, \tag{4.17}$$

corresponding to constants a and b. The (a, b) approximants are known as [61],

$$N_{a,b}(\mathbf{Q}t) = \sum_{j=0}^{a} \frac{(a+b-j)!a!}{(a+b)!j!(a-j)!}(\mathbf{Q}t)^j, \tag{4.18}$$

and

$$D_{a,b}(\mathbf{Q}t) = \sum_{j=0}^{b} \frac{(a+b-j)!b!}{(a+b)!j!(b-j)!}(-\mathbf{Q}t)^j. \tag{4.19}$$

According to [128], when $a = b$ the Padé approximation becomes the *diagonal Padé* approximation method, in which case equation (4.17) and equation (4.18) become

$$R_{a,a} = \frac{N_{a,a}(\mathbf{Q}t)}{N_{a,a}(-\mathbf{Q}t)} \tag{4.20}$$

with,

$$N_{a,a}(\mathbf{Q}t) = \sum_{j=0}^{a} \frac{(2a-j)!a!}{(2a)!j!(a-j)!}(\mathbf{Q}t)^j. \tag{4.21}$$

The diagonal Padé approximation is more stable than other choices, since in Markov chain problems, the eigenvalues of the initial-value problem are found in the left half-plane, causing error in the approximates of $R_{a,b}(\mathbf{Q}t)$ [92]. If $a > b$, cancelation problems can lead to large roundoff errors, and if $a < b$, $D_{a,b}(\mathbf{Q}t)$ may be badly conditioned [92]. Furthermore, a higher-order method is obtained with the same amount of computation, since the same amount of work is needed to compute $R_{aa}(\mathbf{Q}t)$, and this approximation has order $2b > a + b$.

Equation (4.21) can be implemented efficiently as,

$$R_{aa} = \begin{cases} \mathbf{I} + 2\dfrac{(\mathbf{Q}t)\displaystyle\sum_{k=0}^{a/2-1} c_{2k+1}(\mathbf{Q}t)^{2k}}{\displaystyle\sum_{k=0}^{a/2} c_{2k}(\mathbf{Q}t)^{2k} - (\mathbf{Q}t)\sum_{k=0}^{a/2-1} c_{2k+1}(\mathbf{Q}t)^{2k}} & \text{if } a \text{ is even} \\ -\mathbf{I} - 2\dfrac{(\mathbf{Q}t)\displaystyle\sum_{k=0}^{(a-1)/2} c_{2k}(\mathbf{Q}t)^{2k}}{\displaystyle\sum_{k=0}^{(a-1)/2} c_{2k+1}(\mathbf{Q}t)^{2k} - \sum_{k=0}^{(a-1)/2} c_{2k}(\mathbf{Q}t)^{2k}} & \text{if } a \text{ is odd} \end{cases} \tag{4.22}$$

where c_i is defined by,

$$c_0 \equiv 1 \qquad\qquad c_i \equiv c_{i-1}\frac{a+1-i}{i\,(2a+1-i)}, \quad i = 1, 2, \ldots.$$

As described above, a major disadvantage of using the Padé method is that the accuracy of the approximation decreases with distance from the origin and thus degrades with the increasing size of $\|\mathbf{Q}t\|_2$. However, if for a and b are sufficiently large, or if the eigenvalues of $\mathbf{Q}t$ are negative, then the nonsingularity of $D_{a,b}(\mathbf{Q}t)$ is assured [92]. Therefore, Padé approximation can be used if $\|\mathbf{Q}t\|$ is not too large.

Moler and Van Loan [92] perform a comparative analysis of the efficiency of the Padé and Taylor approximants and analyze their relative compatibility with the squaring and scaling method. They find that the combination of the Padé approx-

```
**Require:** Scale $\mathbf{Q}t$ by power of 2 so that $\|\mathbf{Q}t\| < 1/2$
  **Input:** $\mathbf{Q}t$
  **Set:** $b = 6$, $a =$ True, $c = 1/2$
  {Compute Padé approximation for $exp(\mathbf{Q}t)$}
  $X \leftarrow \mathbf{Q}t$
  $\mathbf{E} \leftarrow \mathbf{I} + c\mathbf{Q}t$
  $\mathbf{D} \leftarrow \mathbf{I} - c\mathbf{Q}t$
  **for** $k = 2$ to $b$ **do**
    $c \leftarrow c(b - k + 1)/k(2b - k + 1)$
    $X \leftarrow \mathbf{Q}t\,X$
    $\psi \leftarrow c\,X$
    $\mathbf{E} \leftarrow \mathbf{E} + \psi$
    **if** $a$ is True **then**
      $\mathbf{D} \leftarrow \mathbf{D} + \psi$
    **else**
      $\mathbf{D} \leftarrow \mathbf{D} - \psi$
    **end if**
    Set $a$ to the Boolean conjugate of its current value
  **end for**
  $\mathbf{E} \leftarrow \mathbf{D}/\mathbf{E}$
  {Undo scaling by repeated squaring}
  **for** $k = 1$ to $s$ **do**
    $\mathbf{E} \leftarrow \mathbf{E}^2$
  **end for**
```

Figure 4.8 Algorithm 8: Padé Approximation via squaring and scaling.

imation with repeated squaring and scaling is the most stable and computationally efficient method. Moreover, the MATLAB `expm` command employs the Padé approximation via repeated squaring and scaling. MATLAB's implementation cites [92] and [59], with guidance from Ward [140], as is detailed in Appendix A. Their procedure is demonstrated in Algorithm 8.

4.4.2 Numerical inversion of Laplace transforms

In order to compute the numerical inversion of the Laplace transforms in equation (3.15), a method provided by Abate and Whitt in [5] was chosen. In [5] two inversion algorithms are presented, both variants of the Fourier-series method. The

Fourier-series method is the term used to describe the process of implementing the Fourier series of an associated periodic function to numerically integrate via trapezoidal integration. The error for this algorithm is bounded only partially by using the Poisson summation formula to identify the discretization error associated with the alternating series in the algorithm provided by the trapezoidal rule.

The inverse Laplace transform is defined as,

$$\mathcal{L}^{-1}\left\{\hat{f}(s)\right\} = f(t) = \frac{1}{2\pi i}\int_{a-i\infty}^{a+i\infty} e^{st}\hat{f}(s)\,ds, \tag{4.23}$$

where $\hat{f}(s)$ is the Laplace transform of the function $f(t)$, and $i = \sqrt{-1}$. If $f(t)$ is a real-valued function, the right-hand side of equation (4.23) can be expressed per [52] as

$$\frac{2e^{at}}{\pi}\int_{0}^{\infty} \mathrm{Re}\left[\hat{f}(a+iu)\right]\cos(ut)\,du \tag{4.24}$$

where Re[·] denotes the real part of a complex number.

In practice, explicitly computing equation (4.24) is difficult and often not possible, requiring the use of numerical methods. There is a significant body of work in the literature on various techniques to numerically evaluate equation (4.24) such as [132, 8, 7, 6, 9, 5, 58, 3, 4, 2, 141, 1]. Most of these articles apply the finite fourier cosine transform, which relates directly to the Laplace transform. Dubner and Abate introduced an approximation formula in their seminal paper [52], using this method to approximate equation (4.24) to any desired accuracy according to the following formula,

$$\begin{aligned} f(t) &\approx \sum_{n=0}^{\infty} e^{at} g_n(t) \\ &= \frac{2e^{at}}{T}\left[\frac{1}{2}\mathrm{Re}\left\{\hat{f}(a) + \sum_{k=1}^{\infty}\mathrm{Re}\left\{\hat{f}\left(a+\frac{k\pi i}{T}\right)\right\}\cos\left(\frac{k\pi}{T}t\right)\right\}\right] \end{aligned} \tag{4.25}$$

where $g_n(t)$ is one of an infinite number of even periodic functions, each with period $2T$. Equation (4.25) is actually the the trapezoidal rule [38] applied to equation (4.24), which is the foundation upon which most numerical Laplace transform inversions are implemented.

However, numerous other methods have been implemented both by extending the method of [52] into multiple dimensions and attempting to use other methods such as Laguerre functions [141]. However, methods employing Laguerre functions perform poorly on problems with large values of t, and further problems are introduced by their non-geometric convergence [120].

In this thesis, Laplace transforms are inverted using an algorithm of Abate and Whitt [5], which uses a variant of the Fourier-series method specifically tailored to Laplace transforms of probability distributions. Abate and Whitt [5] present two distinct methods to invert equation (4.24). The first uses the Bromwich integral [116], the Poisson summation formula, and the Euler summation, while the second uses the Post-Widder formula, the Poisson summation formula, and the Stehfest enhancement [2].

Equation (4.24) can be evaluated by the trapezoidal rule with step size $h = \pi/2x$ and $a = A/2x$, yielding the following series,

$$f_h(x) = \frac{e^{A/2}}{2x} \operatorname{Re}[\hat{f}] \left(\frac{A}{2x}\right) + \frac{e^{A/2}}{x} \sum_{k=1}^{\infty} (-1)^k \operatorname{Re}[\hat{f}] \left(\frac{A + 2k\pi i}{2x}\right). \tag{4.26}$$

Since equation (4.26) involves an infinite sum, Abate and Whitt [5] chose the Euler summation acceleration technique due to its simplicity and adequate computational efficiency. Defining $s_n(x)$ as the approximation to $f_h(x)$ in equation (4.26) via Euler approximation techniques, $s_n(x)$ becomes

$$s_n(x) = \frac{e^{A/2}}{2x} \operatorname{Re}[\hat{f}] \left(\frac{A}{2x}\right) + \frac{e^{A/2}}{x} \sum_{k=1}^{n} (-1)^k a_k(x), \tag{4.27}$$

where $a_k(x)$ is defined as

$$a_k(x) = \mathrm{Re}[\hat{f}]\left(\frac{A + 2k\pi i}{2x}\right). \tag{4.28}$$

Since the Euler summation is applied to m terms after an initial n, the full Euler sum approximation is

$$E(m, n, x) = \sum_{k=0}^{m} \binom{m}{k} 2^{-m} s_{n+k}(x), \tag{4.29}$$

and Abate and Whitt [5] recommend the values $m = 11$ and $n = 15$. Equations (4.27), (4.28) and (4.29) were implemented in the MATLAB function `invt.m`, given in Appendix C.

Because the Laplace transform inversion algorithm must be called with each function call in the main availability computation (Algorithms 2 and 3), it is critical to minimize the computational time of the inversion algorithm. The inputs to the inversion procedure are: the function to be evaluated (in this case, the component lifetime distribution function $G_i(x, t)$), the initial probability distribution vector $\boldsymbol{e}_i$ and the current inter-inspection interval τ. Since the computational complexity of this algorithm does not vary with the input size, it is $\mathcal{O}(1)$.

Even with an asymptotic complexity of $\mathcal{O}(1)$, (the algorithm was implemented in `UBASIC` by Abate and Whitt in [5]) it contains over 140 lines of code. In order to reduce this computational expense, quantities that only needed to be computed once were pre-calculated and placed outside of Algorithm 9. In [5], Abate and Whitt recommended values for the constants for the Euler summation acting as an average of the last m partial sums of a binomial probability distribution with parameters m and $p = 0.5$. They applied the Euler summation to m terms after an initial n of equation (4.29) and recommended values of $m = 11$ and of $n = 16$. For notational simplicity and computational savings, they set the quantity 2^{11} in equation (4.29) to

```
Input: $EG$, $z0$, $n\tau$
Load: $\chi$, $\sigma$, $u$, $c$, $\mu$, $h$, $\nu$
$\varsigma \leftarrow G(\chi, \boldsymbol{\alpha}, n\tau, f(\chi))/2$
for $k = 1$ to $\nu$ do
  $\varsigma \leftarrow \varsigma + (-1)^k G(\chi + khi, \boldsymbol{\alpha}, n\tau, f(\chi + khi))$
end for
Store $\varsigma$ as the first element of $\sigma$
for $k = 1$ to 12 do
  $n \leftarrow \nu + k$
  $\sigma(k+1) \leftarrow \sigma(k) + (-1)^n G(\chi + nhi, \boldsymbol{\alpha}, n\tau, f(\chi + nhi))$
end for
Output $\leftarrow u(c \cdot \sigma)/\mu$
```

Figure 4.9 Algorithm 9: Laplace transform inversion algorithm.

μ^{-1} and $A = 8\ln(10)$. The quantity, $(e^{A/2})/x$ in equation (4.27) is assigned to the variable u and the quantity $\chi = A/(2x)$ and $h = \pi/x$.

In this chapter, typical optimization strategies were surveyed. Generalized pattern search was presented as a suitable optimization algorithm, and a specific Matlab implementation was discussed. A direct and improved algorithm for computing $\bar{A}(\tau)$ was presented with a discussion of three different computational improvement strategies applied to the computation of $\bar{A}(\tau)$, with specific emphasis on the matrix exponential and numerical Laplace inversion.

The next chapter illustrates the implementation of the solution procedures presented here. After a description of the overall experiment, the pattern search methodology used to maximize $\bar{A}(\tau)$ will be discussed. Also, the results from the matrix exponential study will be presented. Five different cases will be considered, each with a distinct set of parameters. Primarily characterized by the number of environmental states, the cases were designed to test the broad applicability of each chosen methodology. A particular emphasis is placed upon the computational improvements obtained in each case by the use of the truncation method.

5. Numerical Results

This chapter summarizes and discusses the results of a series of numerical experiments involving the maximization of $\bar{A}$ by selection of an appropriate inter-inspection interval, τ. All numerical computations were performed in MATLAB. The NOMADm implementation of the GPS algorithm was used to solve for the approximate optimum inter-inspection interval. Results are presented for five scenarios involving degradation from environment-dependent linear wear and random shocks.

Two experiments were designed to illustrate the optimization problem and to evaluate particular computational improvements. The first experiment evaluated four different implementations of the matrix exponential algorithm in order to determine the most efficient and stable implementation. The meaning of these terms in the context of this research will be discussed below. The second experiment examined the performance gains provided by the truncation method discussed in chapter 4. The truncation method used only the most suitable matrix exponential algorithm resulting from the previous experiments.

In numerical analysis, the *stability* of an algorithm refers to its accuracy. A given algorithm is numerically *stable* if it produces a good approximation to the true solution [77]. Since solutions produced in this thesis are numerical approximations, stability will refer to the particular capability of an algorithm to arrive at a solution consistently produced by algorithms of known stability. Often a given quantity can be numerically computed in several different ways, all of which are algebraically equivalent, but in practice yield different results because one method is more stable than another. An algorithm will be termed *unstable* if it does not return an answer, due to a undesired halt in code execution. If one algorithm is more *efficient* than another, it more effectively uses computational resources (such as time and storage) in order to perform a given computation [143]. In this research, efficiency is measured by a combination of algorithmic complexity and the elapsed run time required to execute an algorithm.

Both experiments were conducted in the MATLAB computing environment on a Silicon Graphics (SGI) Origin 3900 maintained by the Air Force Aeronautical Systems Center. The Origin 3900 is a large-scale server running the custom SGI unix-based operating system IRIX with over 2048 processors providing a peak computational power of 2.9×10^{12} floating point operations per second (FLOPS). Each processor is a SGI MIPS R16000 with a clock speed of 700 MHz and one gigabyte of dedicated random access memory (RAM). A user is given exclusive access to each allocated processor and its associated RAM for computational use. This is useful when comparing the computational time required by individual runs, as there are no fluctuating background processes interfering with benchmarks, as is the case on a personal computer. The relative machine accuracy for floating point numbers on the Origin 3900 using MATLAB is 2.2204E$-$16. The primary benchmark to measure a given algorithm's performance was the clock-time elapsed during each individual run, denoted by ρ.

Five different cases were constructed, each with a distinct combination of characteristic parameters to fully test the algorithms presented over a wide range of scenarios. These parameters are: the generator matrix for the environment with an associated linear wear rate for each state, the shock distribution and Poisson arrival rate, along with the critical damage threshold for the system. Parameters were chosen in order to demonstrate the optimization algorithms over a wide range of possibilities and are presented in the following sections in order of the associated dimension of the random environment.

In the seven-, ten- and twenty-state cases, the transition rates present in the generator matrix $\mathbf{Q}$ were randomly generated using the MATLAB `rand` function. To create a valid infinitesimal generator matrix, a MATLAB script set the diagonal elements to the negative of the sum of the remaining elements in the row. The

MATLAB `rand` function uses a modified version of Marsaglia's [86] *subtract with borrow* algorithm and can theoretically generate over 2^{1492} values before repeating itself. Other parameters were chosen in order to cover a range of distributions and wear rates along with varying values of the maximum lifetime $\Lambda = x/r_1$. This ratio is of particular importance, since computational requirements are extremely sensitive to Λ when combined with small values of τ.

In order to control and initiate batch runs, a series of scripts in the Perl computer language were produced to prepare the file structure for each run and interface with the MSRC load sharing facility (LSF) to submit jobs. Batch runs generating replications were necessary, since the elapsed clock-time provides the computational basis of comparison between any two algorithms, and there is an inherent noise due to the physical characteristics of an individual processor that can be approximated as random noise present in computational run times [60]. The effect of noise becomes more pronounced as the number of different processors used in the experiment increases. The noise present in computational run times on large-scale servers is due to interactions from an number of processes [60]. Most prominent among these are access to level-3 cache and differences in the characteristics of the various processors performing runs in the server. Since a large-scale server contains a number of different processors, each with a unique actual processing speed, there is an inherent noise in results, even when processors are dedicated to a particular user and all with equal processing speeds.

For these reasons, a statistical analysis was conducted to determine the number of runs required, and the main batch generation script generated 30 jobs for each combination of method and environmental scenario. This number was chosen in order to provide a sufficiently small confidence interval for the computational run time while still working within the budget available for computing resources. After completion of all experiments, a shell script collected all data, which were then processed and analyzed on a personal computer using a custom MATLAB post-processing script.

The first experiment evaluated four different methods to compute the matrix exponential as discussed in chapter 4. These methods are the Taylor series, the method of eigenvalues and eigenvectors (EE), the Uniformization method and the Padé approximation via repeated squaring and scaling (PSS). Each scenario was examined and evaluated for stability and computational efficiency. In order to determine the relative efficiency of the various methods, a series of heteroscedastic, two-tailed t-tests were conducted to test the null hypothesis that there is no statistical difference between the run times of any two given methods. Confidence intervals for ρ were computed at a 0.05 level of significance. After completion of the analysis, the most stable and computationally efficient matrix exponentiation method was found to be the Padé approximation via repeated squaring and scaling.

The second experiment used the results of the first to specifically consider the computational advantage provided by the truncation method. As in the first experiment, 30 runs were conducted for the baseline and truncation method for all five cases. The baseline performed the full, direct computation, while the truncation method trimmed all successive terms smaller than 0.0001. Similarly, a heteroscedastic two-tailed t-test was conducted to ensure any improvement in computational run time due to truncation was statistically significant.

5.1 NOMADm Configuration

This section discusses the particular configuration of NOMADm used to produce the results of the numerical experiments presented in the following sections. The overall optimization problem was set up to work with a main function file, named `availabilityCalc`, which evaluates an iterate τ and returns the corresponding objective $\bar{A}(\tau)$ and cost function values. A file named `availabilityCalc_x0` simply returns the initial point which is the center of the maximum lifetime (Λ) of the component. The feasible region (with respect to bound constraints) for the decision variable is defined in a file named `availabilityCalc_Omega`. In order to reduce computational complexity, all calculations requiring only one evaluation were placed in a separate parameter file `availabilityCalc_Param`, which returns a MATLAB structure whose fields contain all parameters used to characterize a particular scenario, plus any other calculations that may be computed in advance. The use of this file had a dramatic effect on decreasing the overall run time.

The one-dimensional poll directions for this problem were chosen as $\{-1, 1\}$, and the poll set was evaluated only until an improvement is found, as opposed to evaluating the entire set. The initial starting point τ_0 was chosen to be the midpoint of the feasible region $(0, \Lambda]$, which seems reasonable because little information is available concerning the structure of the optimization problem. While most operationally important values of τ are small relative to the maximum lifetime, starting at a very small value of τ significantly increases computational cost unnecessarily. This is because the component maximum lifetime Λ remains fixed, and small values of τ dramatically increase γ and, consequently, the computational time. Since the pattern search method works by forming a mesh which is reduced in size as the algorithm progresses, it is much better to approach the optimal solution and take large steps in the potentially wrong direction where τ values are larger.

As the MADS optimizer implemented by NOMADm is an iterative method, termination was set to occur when the mesh size shrunk below a tolerance of 10^{-4}, based upon some preliminary empirical studies, or when the number of function evaluations exceeded 50,000.

NOMADm's speed of convergence is sensitive to the settings governing mesh control. The initial mesh size was set to $\Delta_0 = \Lambda/4$ as a compromise between a step sufficiently large to quickly approach the optimal τ, but not so large as to overshoot and end with a small mesh in the computationally expensive region where τ is small. The mesh refining factor was set to 0.5, and no mesh coarsening was employed.

The shock distribution parameters, shock arrival rate, and critical damage threshold for each run are presented in exact representations in Table 5.1. Cost

Table 5.1 Key run parameters.

States	Shock Distribution	Shock Mean	λ	x	$\min(\mathbf{r})$	Λ
2	Exp(4)	0.25	0.5	1	0.25	4
5	Er(0.2,8)	40	0.25	100	1	100
7	U(0,10)	5	1	50	2.5	20
10	Er(0.5,4)	8	1	60	0.4	150
20	Gamma(2,4)	8	1	100	0.94	106

parameters were held constant in both experiments and in all five runs. The cost of downtime per unit time was set to 0.5 units per unit time, the cost per-inspection was set to 1 unit and the cost of replacement was set to a fixed 5 units. The generator matrices, linear wear rates and Laplace-Stieltjes transforms for the shock damage magnitude are presented individually in the sections that follow.

5.2 2-State Case

The two-state case, adapted from [67], models a system that transitions between two environment states $S = \{1, 2\}$, each with a distinct rate of linear wear. Environmental transitions occur according to the infinitesimal generator matrix,

$$\mathbf{Q} = \begin{bmatrix} -25/3 & 25/3 \\ 25/3 & -25/3 \end{bmatrix},$$

and the following linear wear rates,

$$\mathbf{r} = \begin{bmatrix} 13/12 & 1/4 \end{bmatrix}.$$

The random damage magnitude due to shocks is assumed to be distributed exponentially with rate 4, and the Laplace-Stieltjes transform of this distribution is given by

$$\tilde{F}_y(u) = \frac{4.0}{4.0 + u}.$$

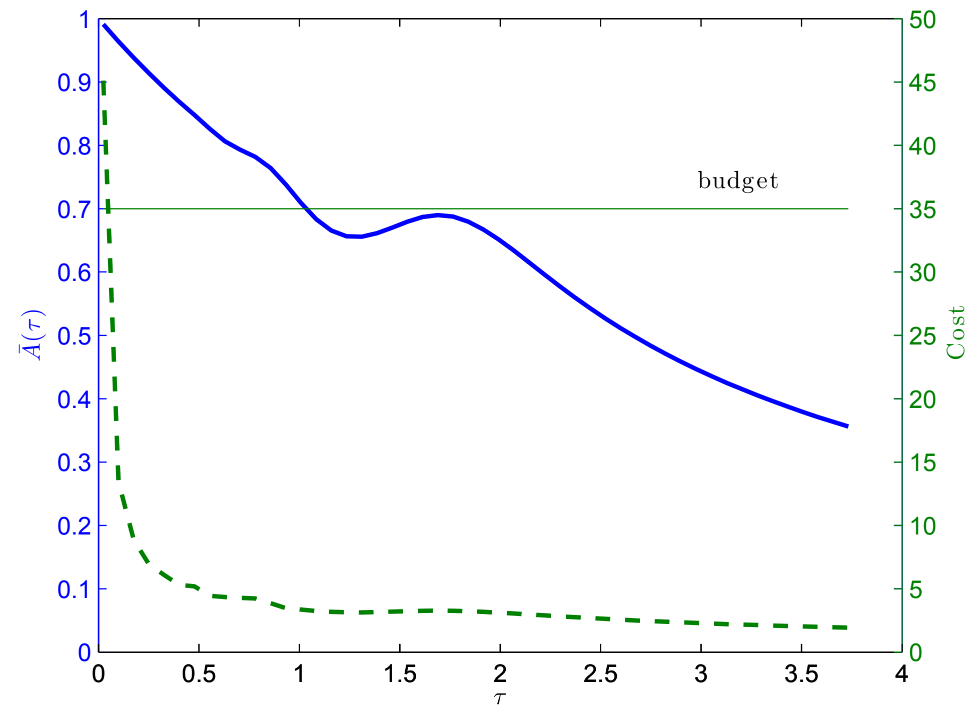

Figure 5.1 Availability (-) and cost (- -) for the 2-state case.

In this example, shocks arrive according to a Poisson process with rate parameter $\lambda = 0.5$. The critical damage threshold level, x, was set to 1.0. From the wear rates presented above, the minimum wear rate is 0.25 and the damage threshold is 1.0 producing a maximum lifetime of 4 time units. The budget is set to 35. A plot of the limiting average availability and cost over several values of τ is presented in Figure 5.1.

The results from the matrix exponentiation experiment are displayed in Table 5.2. All four matrix exponentiation methods produce the approximate optimal inter-inspection time of $\tau^{\star} = 1.69$, with the Taylor series method slightly out of agreement with the others. The Taylor series method did result in 24 function calls to reach an approximate optimal solution while the other three only required 23 which suggests instability with the Taylor series method as discussed in chapter 4.

Table 5.3 displays the p-values of the methods relative to each other. From Table 5.2, we see that the EE method requires the smallest run time, followed by the Uniformization method. This must be stated cautiously since the EE method uses a quickly executing compiled LAPACK function to compute the eigenvectors and eigenvalues. According to Table 5.3 the EE method's computational advantage is statistically significant at a high level of significance (at least on the order of 10^{-10}). Also notable, is that the Taylor series is the statistically the slowest method.

Table 5.2 Two-state matrix exponentiation results.

	Uniformization	EE	Taylor	PSS
$\tau^\star$	1.6971	1.6942	1.6877	1.6942
$\bar{A}(\tau^\star)$	0.68989	0.68980	0.68957	0.68980
ρ	12.065	11.960	13.613	12.094
σ_ρ	0.048449	0.048628	0.032390	0.081434

Table 5.3 Two-state matrix exponential p-value comparison.

	EE	Taylor	PSS
Uniformization	1.73E−11	4.96E−68	1.14E−10
EE		2.75E−69	7.13E−10
Taylor			9.93E−47

For the 2-state case, there is no statistical difference between the computational run times when contrasting the baseline and truncation method with an associate p-value from the t-test of 0.15. Both methods found the approximate optimal τ in around 12 seconds. This is expected, since Λ and the dimension of $\mathbf{Q}$ are both small, the gain provided by the truncation method is not significant. It is also important to note that the only measure showing a measurable difference among the thirty runs is the computational run time. There is no detectible variance between the values of $\tau^\star$ or $\bar{A}(\tau^\star)$ in this or any of the following cases. The mean run time (ρ) for the baseline method was 12.11 seconds in the interval from 12.09 to 12.12 seconds. For the truncation method ρ was 12.09 seconds in the interval from 12.07 to 12.11 seconds.

Table 5.4 Two-state truncation method and baseline comparison.

	Full Sum	Truncated Sum
$\tau^\star$	1.694213867	1.694213867
$\bar{A}(\tau^\star)$	0.689801977	0.689801977
ρ	12.10796217	12.09131087
σ_ρ	0.042446646	0.046830392

5.3 5-State Case

In the 5-state case, which is also adapted from [67], shocks arrive according to a Poisson process with a rate parameter, $\lambda = 0.25$. The critical damage threshold level is set at $x = 100$, and the overall budget for inspections, down-time, and replacements is 2 units. The generator matrix is,

$$\mathbf{Q} = \begin{bmatrix} -0.500 & 0.125 & 0.125 & 0.125 & 0.125 \\ 0.400 & -2.000 & 0.400 & 0.600 & 0.600 \\ 0.025 & 0.025 & -0.100 & 0.025 & 0.025 \\ 0.050 & 0.050 & 0.050 & -0.200 & 0.050 \\ 1.500 & 1.000 & 1.000 & 1.500 & -5.000 \end{bmatrix}.$$

The condition number of $\mathbf{Q}$ is on the order of 10^{17} which could cause stability problems with the matrix exponential calculations using either the Taylor series or the EE method. The vector of linear wear rates corresponding to each state is

$$\mathbf{r} = [\ 1 \quad 2 \quad 3 \quad 4 \quad 10\ ].$$

Since the minimum wear rate is one and $x = 100$ the system lifetime of 100 is much longer than in the two-state case. This causes a larger computational burden, due to the much larger value of γ when τ is small. The shock magnitudes are Erlang distributed with parameters 0.2 and 8, such that the corresponding Laplace-Stieltjes

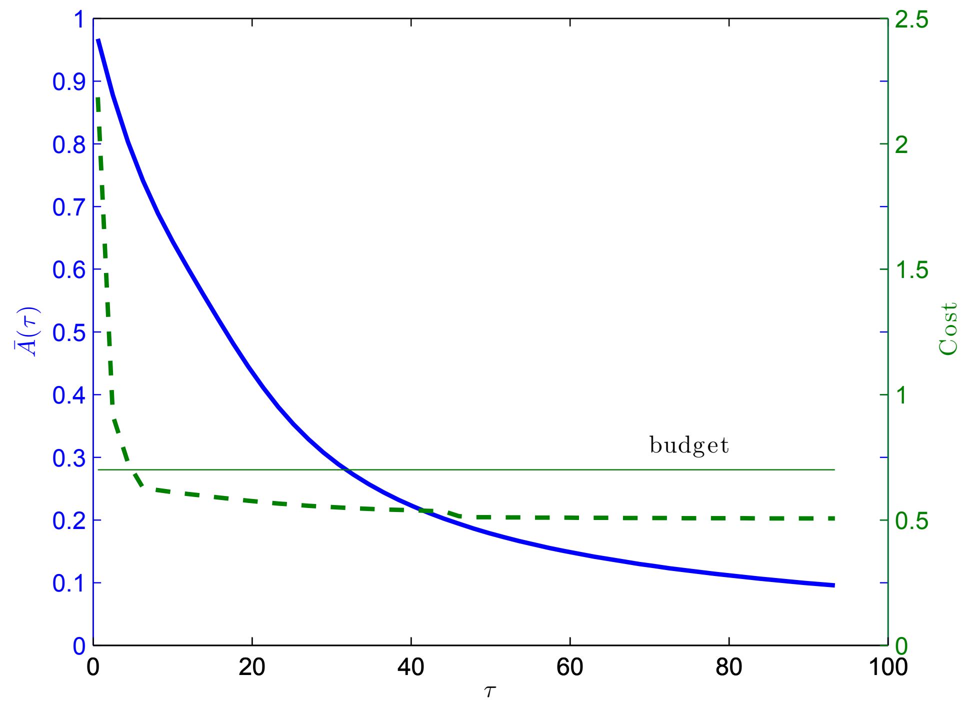

Figure 5.2 Availability (-) and cost (- -) for the 5-state case.

transform of the shock distribution is

$$\tilde{F}_y(u) = \left(\frac{0.2}{0.2+u}\right)^8 .$$

A plot of the limiting average availability and cost over time for this case is presented in Figure 5.2.

Table 5.5 Five-state matrix exponentiation results.

	Uniformization	EE	Taylor	PSS
$\tau^\star$	5.7961	5.7961	50.154	5.7961
$\bar{A}(\tau^\star)$	0.75477	0.75580	0.20703	0.75580
ρ	867.64	852.74	85.81	865.33
σ_ρ	4.34	5.41	0.672	10.2

Table 5.5 displays the five-state case matrix exponential results. It is immediately clear that the Taylor series method, while much faster, is producing vastly different results which implies instability and consequently the Taylor series method

Table 5.6 5-state p-values for matrix exponentiation performance tests.

	EE	Taylor	PSS
Uniformization	1.1E−16	7.0E−70	0.26
EE		8.4E−66	3.4E−7
Taylor			7.7E−57

cannot be considered a viable option for this case. The three remaining options required 73 function evaluations compared to the 27 required by the Taylor series method.

The p-values resulting from comparative t-tests between the run times of the various matrix exponentiation methods are presented in Table 5.6. The fastest method was EE, with the slowest Uniformization, even though all three were fairly close with regard to ρ. There was no statistical difference between the run times of the Uniformization and PSS method, while the difference between all other methods was statistically significant. Again, as in the 2-state case, all three of the remaining methods produced very similar results for $\tau^\star$, but the Uniformization method produced a slightly different value for $\bar{A}(\tau^\star)$. This suggests the Uniformization method may be producing some slightly inaccurate solutions. Based upon this analysis, the EE method is the most appropriate method for the five state case.

Results from the truncation experiment are listed in 5.7. The increased state dimension and larger value of Λ resulted in more terms being truncated, with an associated significant decrease in run time. The truncation method provided a 17% improvement over the baseline method. From conducting a t-test, this difference was highly statistically significant with a p-value on the order of 10^{-59}. For the baseline, the mean of ρ was found to be 859.62 sec in the confidence interval from 857.07 to 862.18 sec. For the truncation method the mean of ρ was found to be 715.67 sec in the confidence interval from 713.81 to 717.53 sec.

Table 5.7 5-state truncation method and baseline comparison.

	Baseline Method	Truncation Method
$\tau^\star$	5.796142578	5.796142578
$\bar{A}(\tau^\star)$	0.755800236	0.755800236
ρ	859.63	715.67
σ_ρ	7.15	5.19

5.4 7-State Case

The seven-state case models an environment with a moderate number of states. The damage threshold is $x = 50$ units and the Poisson shock arrival rate is $\lambda = 1$. The budget was set to 3.5 units with the same costs present in the other cases. The generator matrix is,

$$\mathbf{Q} = \begin{bmatrix} -2.85 & 0.55 & 0.88 & 0.14 & 0.47 & 0.43 & 0.38 \\ 0.73 & -1.99 & 0.17 & 0.01 & 0.06 & 0.23 & 0.78 \\ 0.31 & 0.69 & -4.15 & 0.89 & 0.99 & 0.58 & 0.68 \\ 0.84 & 0.62 & 0.27 & -3.54 & 0.58 & 0.76 & 0.46 \\ 0.57 & 0.79 & 0.25 & 0.30 & -3.01 & 0.53 & 0.57 \\ 0.37 & 0.96 & 0.88 & 0.66 & 0.52 & -4.17 & 0.79 \\ 0.70 & 0.52 & 0.74 & 0.28 & 0.33 & 0.21 & -2.79 \end{bmatrix}$$

with wear rates increasing linearly,

$$\mathbf{r} = \begin{bmatrix} 2.5 & 3.5 & 4.5 & 5.5 & 6.5 & 7.5 & 8.5 \end{bmatrix}.$$

With a minimum wear rate of 2.5 and $x = 50$, $\Lambda = 20$ is moderately sized. The shock magnitude distribution is uniform over the interval [0,10] with the corresponding Laplace-Stieltjes transform,

$$\tilde{F}_y(u) = \frac{e^{-10u} - 1}{10\,u}.$$

A plot of the approximate limiting average availability and cost over time for this case is presented in Figure 5.3.

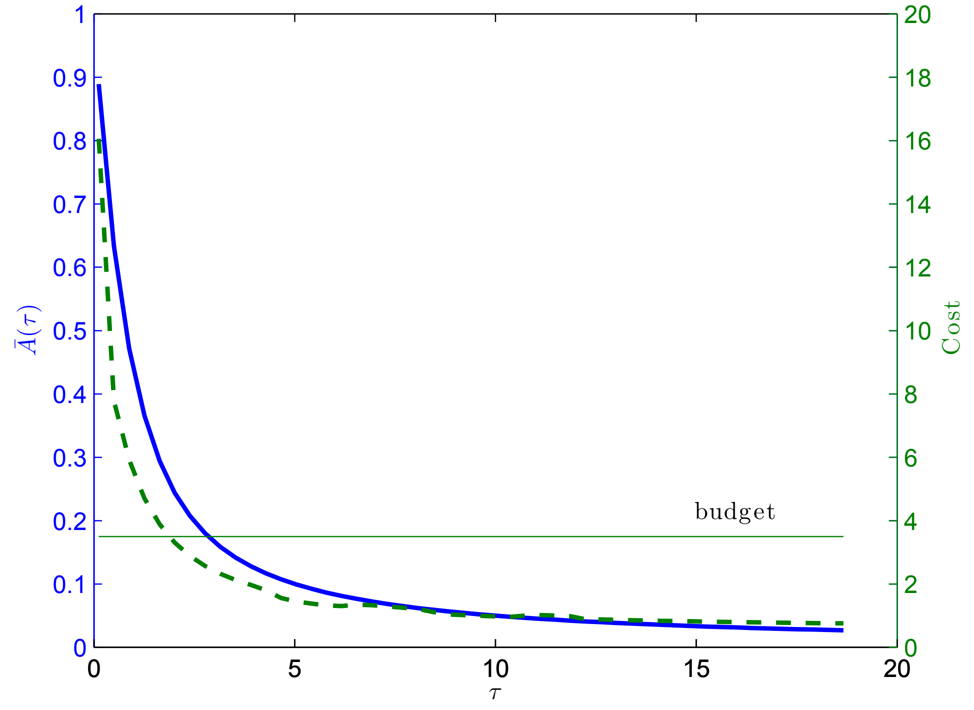

Figure 5.3 Availability (-) and cost (- -) for the 7-state case.

The results from the matrix exponentiation experiment are displayed in Table 5.8. Most notably, the EE method was completely unstable and consistently caused an exception in MATLAB for all thirty runs. Even though $\mathbf{Q}$ has a condition number much smaller (on the order of 10^5) than the first two cases, it has two imaginary eigenvalues which cause the errors. While the Uniformization and PSS methods produced the same $\tau^\star$ to machine precision, the Taylor series method produced a slightly different value. Furthermore, NOMADm performed one less iteration with the Taylor series method implemented. Uniformization and PSS had fairly close, but still significantly different, run times with Uniformization as the fastest method. The Taylor series method was significantly slower. The p-values for these comparisons are shown in Table 5.9. Based on this analysis, Uniformization is the most suitable matrix exponentiation method for this case.

Table 5.8 7-state matrix exponentiation results.

	Uniformization	EE	Taylor	PSS
$\tau^\star$	1.864135742	N/A	1.866210938	1.864135742
$\bar{A}(\tau^\star)$	0.261806575	N/A	0.261735112	0.26180678
ρ	958.1941171	N/A	1028.039258	964.0604266
σ_ρ	6.407937487	N/A	4.518813375	8.108105053

Table 5.9 7-state p-values for matrix exponentiation performance tests.

	Taylor	PSS
Uniformization	3.5E−45	0.002969106
Taylor		5.9E−36

The truncation method showed a 52% improvement as presented in Table 5.10. This difference was clearly statistically significant, with an associated p-value from the comparative t-test on the order of 10^{-47}. The mean value of ρ for the baseline was 966.91 seconds in the confidence interval of [961.40,972.43] seconds and the mean ρ-value for the truncation method was 465.15 seconds in the interval [464.27,466.03].

This improvement is due to the relatively large Λ and increased state size. As the state size grows, the number of computations of equation (3.7) grows with the square of the increase, and any gain made by the truncation method is magnified with each computation.

5.5 10-State Case

The 10-state case serves as a more computationally intensive challenge with which to test the various computational improvements. The shock arrival rate is

Table 5.10 7-state truncation method and baseline comparison.

	Baseline Method	Truncation Method
$\tau^\star$	1.864135742	1.864135742
$\bar{A}(\tau^\star)$	0.26180678	0.261806778
ρ	966.9130686	465.1510443
σ_ρ	15.42015957	2.459362772

$\lambda = 1$, and the critical damage threshold is $x = 60$. The total budget was set to 2 units. The following generator of dimension 10 was constructed,

$$\mathbf{Q} = \begin{bmatrix} -3.86 & 0.06 & 0.49 & 0.51 & 0.07 & 0.49 & 0.41 & 0.93 & 0.37 & 0.53 \\ 0.39 & -3.37 & 0.04 & 0.45 & 0.38 & 0.50 & 0.56 & 0.26 & 0.25 & 0.55 \\ 0.25 & 0.46 & -3.93 & 0.33 & 0.37 & 0.84 & 0.27 & 0.20 & 0.92 & 0.28 \\ 0.35 & 0.86 & 0.33 & -4.67 & 0.48 & 0.81 & 0.78 & 0.05 & 0.63 & 0.37 \\ 0.74 & 0.86 & 0.90 & 0.89 & -6.18 & 0.86 & 0.39 & 0.61 & 0.88 & 0.06 \\ 0.65 & 0.47 & 0.31 & 0.76 & 0.34 & -4.30 & 0.03 & 0.55 & 0.64 & 0.54 \\ 0.94 & 0.79 & 0.25 & 0.88 & 0.25 & 0.57 & -5.41 & 0.10 & 0.80 & 0.84 \\ 0.83 & 0.66 & 0.43 & 0.46 & 0.58 & 0.61 & 0.56 & -4.71 & 0.44 & 0.15 \\ 0.47 & 0.00 & 0.84 & 0.80 & 0.52 & 0.10 & 0.20 & 0.44 & -3.55 & 0.17 \\ 0.63 & 0.13 & 0.18 & 0.13 & 0.16 & 0.16 & 0.09 & 0.07 & 0.10 & -1.65 \end{bmatrix}.$$

The linear wear rates were designed to represent environmental states whose damage rates are increasing in a logarithmic fashion. The vector of rates is

$$\mathbf{r} = \begin{bmatrix} 0.400 & 0.903 & 1.431 & 1.806 & 2.097 & 2.335 & 2.535 & 2.709 & 2.863 & 3.000 \end{bmatrix},$$

with the minimum linear wear rate of 0.400 and $x = 60$, the maximum lifetime is $\Lambda = 150$.

The shock magnitude is Erlang distributed with a Laplace-Stieltjes transform of

$$\tilde{F}_y(u) = \left(\frac{0.5}{0.5 + u}\right)^4.$$

A plot of the approximate limiting average availability and cost over various values of τ is presented in Figure 5.4.

The results of the matrix exponentiation experiment for the 10-state were revealing. As in the 7-state case, some of the eigenvalues of $\mathbf{Q}$ are imaginary which

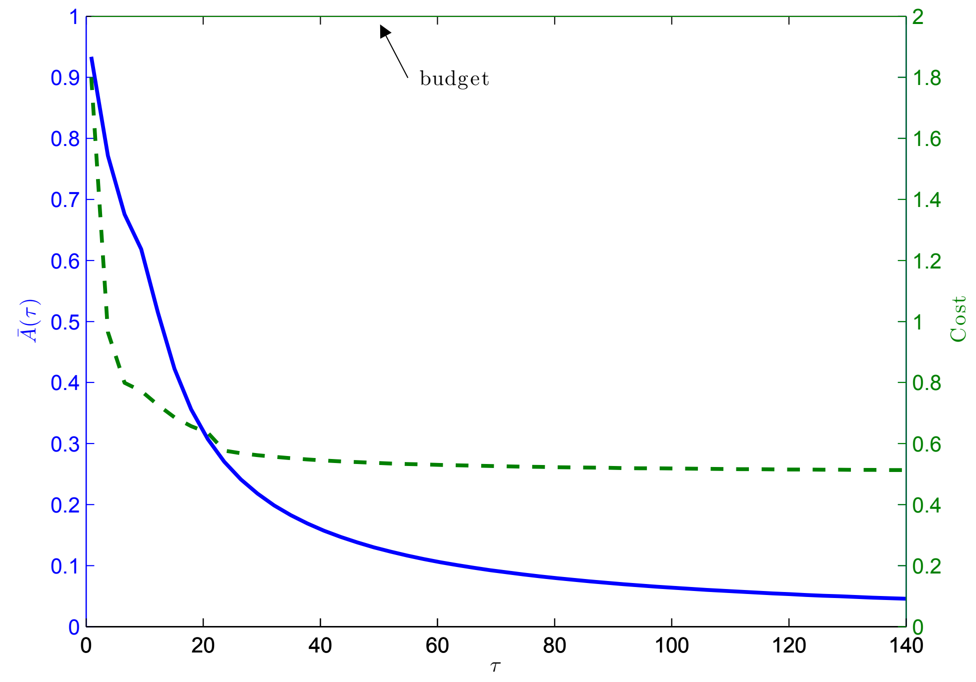

Figure 5.4 Availability (-) and cost (- -) for the 10-state case.

cause the EE method to fail. The Uniformization and Taylor series method both did not provide solutions in the 10-state case, demonstrating instability due to cumulative round off error and machine tolerance limits as discussed in chapter 4. Only the PSS method converged with a $\tau^{\star}$ of 0.75183 and an associated $\bar{A}(\tau^{\star})$ of 0.94437. The average run time was 31574.84 sec with standard deviation of 166.32 sec.

The results of comparing the truncation to the baseline for the 10-state case are presented in Table 5.11. The truncation method requires 67% less computation than the baseline and is significant with a p-value on the order of 10^{-71}. The mean value of ρ for the 10-state case is 31,764.53 seconds, with an associate confidence interval of [31,675.62,31853.43]. The ρ for the truncation method has a mean of 10,637.64 seconds in the confidence interval of [10602.81,10672.47].

Table 5.11 10-state truncation and baseline comparison.

	Baseline Method	Truncation Method
$\tau^\star$	0.751831055	0.751831055
$\bar{A}(\tau^\star)$	0.944372809	0.944372809
ρ	31764.53	10637.64
$\rho_\star$	248.44	97.34

5.6 20-State Case

The twenty-state case was developed to stress the algorithmic improvements presented in this thesis and model the state-size of a comprehensive, realistic, scenario. The critical damage threshold is $x = 100$ and the total maintenance budget is 1.2 units. The 20-dimension generator for this case is displayed in Tables 5.12 and 5.13.

Table 5.12 Twenty-state generator matrix (columns 1 through 10).

$$\left[\begin{array}{rrrrrrrrrrl}
-10.07 & 0.07 & 0.60 & 0.30 & 0.56 & 0.02 & 0.81 & 0.37 & 0.52 & 0.95 & \ldots \\
0.72 & -9.37 & 0.34 & 0.46 & 0.70 & 0.40 & 0.70 & 0.72 & 0.14 & 0.62 & \ldots \\
0.84 & 0.45 & -10.38 & 0.50 & 0.29 & 0.80 & 0.44 & 0.29 & 0.32 & 0.46 & \ldots \\
0.82 & 0.21 & 0.12 & -10.01 & 0.99 & 0.16 & 0.47 & 0.79 & 0.86 & 0.65 & \ldots \\
0.54 & 0.74 & 0.52 & 0.94 & -10.04 & 0.31 & 0.88 & 0.21 & 0.43 & 0.98 & \ldots \\
0.36 & 0.58 & 1.00 & 0.65 & 0.50 & -9.76 & 0.65 & 0.43 & 0.06 & 0.42 & \ldots \\
0.50 & 0.79 & 0.31 & 0.95 & 0.43 & 0.59 & -10.01 & 0.63 & 0.89 & 0.98 & \ldots \\
0.33 & 0.08 & 0.51 & 0.46 & 0.52 & 0.05 & 0.21 & -8.55 & 0.26 & 0.52 & \ldots \\
0.23 & 0.38 & 0.82 & 0.27 & 0.05 & 0.87 & 0.13 & 0.25 & -8.58 & 0.54 & \ldots \\
0.60 & 0.20 & 0.30 & 0.07 & 0.96 & 0.73 & 0.54 & 0.66 & 0.22 & -8.86 & \ldots \\
0.05 & 0.65 & 0.71 & 0.19 & 0.07 & 0.02 & 0.84 & 0.13 & 0.59 & 0.85 & \ldots \\
0.74 & 0.12 & 0.87 & 0.35 & 0.51 & 0.74 & 0.24 & 0.51 & 0.56 & 0.55 & \ldots \\
0.49 & 0.51 & 0.66 & 0.91 & 0.71 & 0.29 & 0.02 & 0.27 & 0.87 & 0.82 & \ldots \\
0.99 & 0.11 & 0.86 & 0.01 & 0.62 & 0.76 & 0.15 & 0.59 & 0.22 & 0.49 & \ldots \\
0.98 & 0.03 & 0.93 & 0.99 & 0.23 & 0.10 & 0.48 & 0.27 & 0.30 & 0.12 & \ldots \\
0.14 & 0.05 & 0.94 & 0.17 & 0.63 & 0.23 & 0.88 & 0.78 & 0.59 & 0.14 & \ldots \\
0.93 & 0.18 & 0.67 & 0.40 & 0.15 & 0.49 & 0.86 & 0.15 & 0.90 & 0.78 & \ldots \\
0.02 & 0.51 & 0.60 & 0.40 & 0.31 & 0.87 & 1.00 & 0.77 & 0.52 & 0.27 & \ldots \\
0.73 & 0.65 & 0.40 & 0.79 & 0.65 & 0.58 & 0.76 & 0.52 & 0.67 & 0.56 & \ldots \\
0.73 & 0.51 & 0.13 & 0.51 & 0.84 & 0.42 & 0.89 & 0.59 & 0.68 & 0.14 & \ldots
\end{array}\right.$$

Table 5.13 Twenty-state generator matrix (columns 11 through 20).

...	0.42	0.68	0.69	0.56	0.74	0.79	0.34	0.51	0.83	0.30
...	0.06	0.86	0.97	0.69	0.49	0.33	0.02	0.05	0.48	0.63
...	0.80	0.10	0.28	0.76	0.65	0.78	0.77	0.99	0.34	0.53
...	0.56	0.39	0.17	0.82	0.04	0.78	0.46	0.31	0.54	0.87
...	0.77	0.38	0.92	0.53	0.33	0.10	0.04	0.26	0.56	0.58
...	1.00	0.52	0.26	0.34	0.43	0.33	0.10	0.61	0.93	0.60
...	0.09	0.89	0.38	0.07	0.10	0.26	0.96	0.45	0.27	0.49
...	0.72	0.64	0.64	0.26	0.68	0.39	0.61	0.42	0.30	0.94
...	1.00	0.20	0.32	0.99	0.03	0.90	0.25	0.41	0.61	0.35
...	0.11	0.07	0.19	0.39	0.32	0.77	0.93	0.71	0.89	0.20
...	−7.55	0.80	0.23	0.79	0.05	0.24	0.43	0.10	0.71	0.10
...	0.13	−8.53	0.22	0.87	0.37	0.45	0.05	0.38	0.20	0.65
...	0.70	0.89	−10.25	0.09	0.38	0.27	0.90	0.75	0.11	0.61
...	0.31	0.99	0.12	−9.65	0.34	0.42	0.98	0.15	0.74	0.81
...	0.06	0.40	0.54	0.84	−9.50	0.70	0.66	0.66	0.76	0.46
...	0.17	0.34	0.69	0.47	0.24	−7.80	0.73	0.24	0.28	0.09
...	0.19	0.32	0.72	0.60	0.08	0.83	−9.93	0.73	0.85	0.10
...	0.67	0.36	0.94	0.21	0.30	0.40	0.69	−10.34	0.62	0.87
...	0.39	0.62	0.01	0.52	0.18	0.65	0.36	0.47	−9.55	0.06
...	0.07	0.60	0.58	0.16	0.72	0.15	0.40	0.86	0.95	−9.91

The rates of linear wear were chosen to increase quadratically with the dimension of $\mathbf{Q}$, as often occurs when environmental effects compound to create damage. The following linear wear rates (to four decimal positions) were used: 0.6444, 1.9778, 4.2000, 7.3111, 11.3111, 16.2000, 21.9778, 28.6444, 36.2000, 44.6444, 53.9778, 64.2000, 75.3111, 87.3111, 100.2000, 113.9778, 128.6444, 144.2000, 160.6444, and 177.9778. Since the minimum linear wear rate is 17/18 and x is 100, Λ is 1800/17.

Shocks are gamma distributed with an arrival rate of $\lambda = 1$ and a Laplace-Stieltjes transform,

$$\tilde{F}_y(u) = (1 + 2u)^4.$$

A plot of the limiting average availability and cost over time for this case is presented in Figure 5.5.

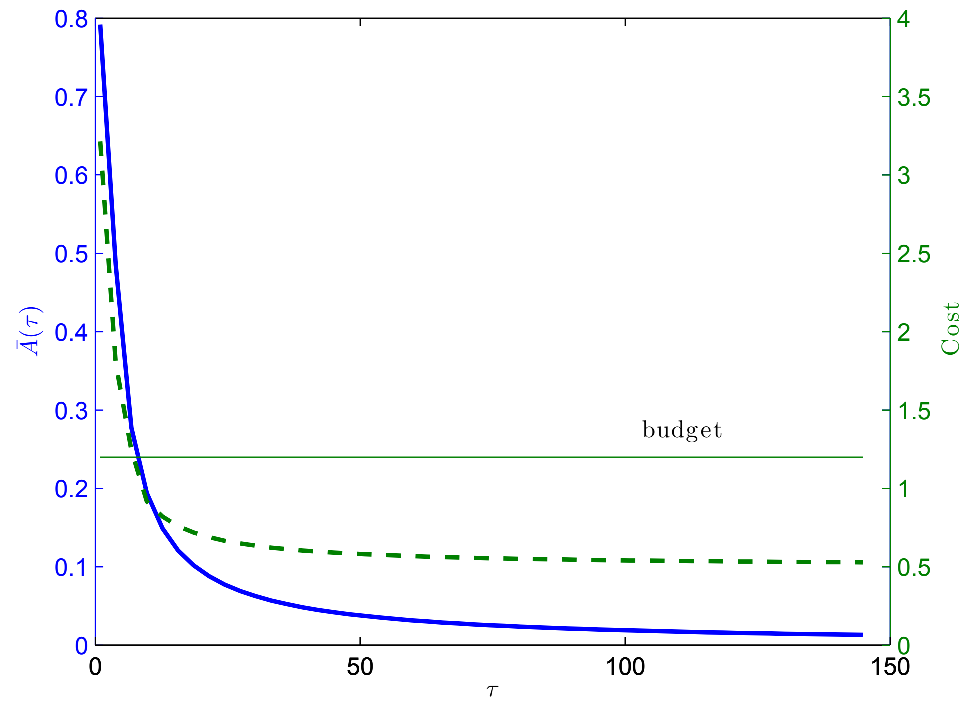

Figure 5.5 Availability (-) and cost (- -) for the 20-state case.

The results from the 20-state matrix exponential experiment show the computational burden incurred by a generator of dimension 20. Again, the condition number of $\mathbf{Q}$ is on the order of 10^{16}; however, the real problem with the EE method is the existence of imaginary eigenvalues that halt execution. The Taylor series method, fails because of excessive round-off error returning an excessively large value for `expm`$(\mathbf{Q}t)$ (the norm of which is on the order of 10^{51}). This large number propagates in the computations until the stationary probability distribution is calculated, which requires a matrix division involving $\hat{P}$. In this case, $\hat{P}$ is badly scaled with a condition number of 10^{48}, resulting in false values for the stationary transition probabilities. The Uniformization method fails for similar reasons. This analysis shows that the PSS method is best suited for the 20-state case.

The results from the comparison experiment between truncation and the baseline for the 20-state case are presented in Table 5.14. Overall, there is a significant decrease (with a p-value on the order of 10^{-78}) in computational cost by 75%. The

Table 5.14 20-state truncation method and baseline comparison.

	Baseline Method	Truncation Method
$\tau^\star$	6.990850031	6.990850031
$\bar{A}(\tau^\star)$	0.270266593	0.270266593
ρ	39367.25	9859.15
σ_ρ	128.39	32.47

number of function calls was 99 in both cases. The mean value of ρ for the baseline case was 39367.25 seconds in the confidence interval of [39,321.31,39413.20] while the mean ρ value using the truncation method was 9859.15 seconds in the interval [9,847.53,9,870.77].

5.7 Summary

The numerical experiments on the matrix exponential computation demonstrated the relative effectiveness of four different implementations. As a secondary benefit, agreement in final solutions provided by the different matrix exponential methods provided confidence in the results of the optimization. An analysis of the results over all cases showed the Taylor series implementation was not sufficiently stable and demonstrated, for this particular problem, that there is no advantage in any of the five cases from implementing the Taylor series. Also, the possibility of imaginary eigenvalues demonstrated that the EE method was also unsuitable, even though it performed the fastest in the 2- and 5-state cases. If, however, the generator can be shown to be Hermitian (i.e. to have all real eigenvalues), then the EE method may be an excellent choice.

The two remaining methods, Uniformization and PSS were considered on the basis of run time comparisons with the Uniformization method faster in the 7-state and the PSS method faster in 5-state case. There was no statistical difference in the 2-state case. In the 10- and 20-state cases, the Uniformization method demonstrated some instability. Therefore, based on the analysis above and the recommendations of [92], the PSS method is to be concluded the best option for this optimization problem. This was also the conclusion of the MathWorks engineering staff in their construction of the `expm` built-in function.

The numerical experiments contrasting the truncation method to the baseline show that the truncation method is very effective with savings proportional to Λ, the dimension of the generator, and the budget. However, each characterizing parameter has an effect on the run time. There is a general trend, in that the truncation method performs better as its associated gains can be magnified through increased state size and increased maximum system lifetimes. This matches the complexity analysis in that the algorithmic complexity is $\mathcal{O}(l^2\gamma)$ with γ directly proportional to Λ for a fixed τ.

Overall, both experiments demonstrated an effective method for maximizing limiting average availability through selecting an appropriate inter-inspection interval τ. The truncation method enables the approximate optimal $\tau^\star$ to be computed within a reasonable amount of time. However, run times are still considerable when maximum lifetimes (Λ) and the dimension of $\mathbf{Q}$ are large. Research into additional computational techniques could further reduce this run time, but the development of a purely analytical technique could provide the true optimal $\tau^\star$ with almost no computational burden. This and other possible extensions to this research are discussed in the next chapter.

6. Conclusions and Future Research

The goal of this research was to present a methodology to determine the approximate optimum inter-inspection duration (τ) that maximizes the limiting average availability of a system while keeping the downtime, inspection and replacement costs within an arbitrary budget for a system with deterioration due to environment-dependent wear and random shocks. In order to be useful, this methodology must provide consistent and accurate results in a reasonable amount of time. To this end, a secondary goal of the research was to improve the computational performance of the chosen methodology.

An optimization methodology using generalized pattern search was chosen and implemented to compute the availability measure $\bar{A}(\tau)$. Moreover, a numerical method, the *truncation method*, was developed which provided computational savings most directly proportional to the dimension of $\mathbf{Q}$ and the component's maximum lifetime. The budget and other model parameters also impact the computational savings of the truncation method. Several standard methods were employed to improve computational improvement. These included: vectorization, preprocessing, and choosing the correct MATLAB operators. In order to properly implement the third method, a study of the matrix exponential computation was conducted to determine the most proper method for this research. The results of this implementation were presented and discussed using five different cases, each with a distinct set of parameters. The parameters were chosen in order to illustrate the means by which to compute the approximate optimal inter-inspection duration and to evaluate the associated performance of the optimization methodology over a wide-range of potential problems. These numerical experiments demonstrated that the truncation method presents significant potential for performance gains over the full-sum case with no loss of solution quality (within the tolerance machine precision).

It is recommended that the NOMADm pattern search code be used, along with the strategies discussed to enhance computational performance of the objective function including the truncation method. NOMADm was observed to be a capable implementation of generalized pattern search. Moreover, NOMADm is distributed without cost under the GNU General Public License and is available direclty from the author. Other recommendations include using the specific one-dimensional Laplace transform inversion algorithm provided by Abate and Whitt [5] and the matrix exponential should be computed via the Padé approximation using repeated squaring and scaling.

Although this research presents a complete methodology considering degradation due to environment-dependent wear and random shocks, much work remains. For instance, to compute an approximate value for $\tau^\star$, the various parameters describing the environment (shock magnitudes and arrival rate, environment transition rates, etc.) must be known. When these parameters are unknown, statistical estimation procedures must be employed. A sensitivity analysis could reveal the most critical parameters and provide insight into the impact that various environmental parameters have on the limiting average availability and reliability measures. Further techniques available to operations research analysts could determine the optimal allocation of resources to impact the environmental parameters to maximize long run availability or minimize cost.

Additional computational improvement techniques remain to be tested for computing $\bar{A}(\tau)$. In particular, since the primary computational expense when computing $\bar{A}(\tau)$ is the population of the transition probability matrix, the application of current research [124] in compressing transition probability matrices could prove useful. Moreover, while all computations in this thesis were performed in MATLAB, implementing the methodology in a compiled language such as `C++` or `FORTRAN` would undoubtedly improve the overall run time by a significant margin. Further research comparing the results of this thesis to meta-heuristic methods could yield computationally fast solutions that also provide approximate optimal solutions.

Additional work on the analytic model could prove fruitful. If a closed-form expression can be found for $\bar{A}(\tau)$ in the time domain, the true optimal point could be found in seconds. A pure analytical solution would also provide considerable insight into the system dynamics. Also, if a mathematically rigorous solution methodology can be developed to optimize the objective function in the complex domain, numerical inversion would not be necessary, and the solution could consequently be computed much faster and with greater precision. Additional insight on the dynamics of the stochastic model could yield a maintenance policy more effective than the deterministic inspect-and-replace policy presented in this thesis.

There is also great potential for applying this methodology to an actual optimal maintenance setting. Much could be learned regarding the applicability of the model from determining environmental transition rates, fitting distributions and parameters and comparing empirical experiments, simulation and calculated reliability and availability measures. Moreover, since the most restrictive assumption is that of a Markovian environment, there is a great deal of value in using the method of Cox [48], who used phase-type distributions to model to approximate semi-Markov environment as a Markovian one.

Appendix A. Availability Calculation

```
function [fx,cx] = availabilityCalc(tau);
%availabilityCalc: function to calculate the availability of a Markovian shock
% and wear model as described in Kharoufeh, et al. (2006).
%
%   Syntax:
%      fx = availabilityCalc(tau);
%     [fx,cx] = availabilityCalc(tau);
%
%   Description:
%   This Matlab program is used to compute conditional distributions and
%   conditional expectations of random quantities from the Markovian
%   shock and wear model in Kharoufeh, et al. (2006).
%
%   References: Kharoufeh, J., Finkelstein, D., and D. Mixon (2006)
%             Availability of inspected systems subject to Markovian wear
%             and shocks.  Technical report. Department of Operational
%             Sciences. Air Force Institute of Technology.
%
%   Calls: invt, unif, expm, ceil, min, ones, zeros, sum, CalcLoopCnt,
%          getappdata, num2str, eye, disp, floor
%
%****************************************************************************
%   Copyright (c) 2006 by Timothy B. Booher
% ---------------------------------------------------------------------------
%   Originally created, 2005.
%   Last modified, 1 March 2006
%
%   Author information:
%   Timothy B. Booher, Capt, USAF
%   Air Force Institute of Technology
%   Department of Operational Sciences
%   2950 Hobson Way
%   Wright-Patterson AFB, OH 45433
%   Timothy.Booher@afit.edu
%****************************************************************************
%
% ---------------------------------------------------------------------------
% VARIABLES (only for the availabilityCalc function):
%*INPUT*VARIABLES***********************************************************
%  tau = the inter-inspection interval and decision variable
%*FUNCTION*VARIABLES********************************************************
```

```
%    l = overall dimension (number of states in Markovian environment)
%    X = marginal PMF of R_1
%    Q = infinitesimal generator matrix for random environment
%    r = vector of wear rates for the Markovian environment
%    lambda = shock arrival rate
%    lim = the failure threshold (x in Kharoufeh et al.)
%    ER_1 = the expected value (unconditional) of the time to the first
%           replacement epoch
%    EF_1 = the expected value (unconditional) of the first failure
%           epoch
%    Pt = transition probability function
%    P = transition probability matrix (of the embedded DTMC)
%    p = stationary distribution, p, of P
%    z0 = e_i, vector of ones with an 1 in the ith position
%    M = ratio of T_max to tau (must be less than one) the ceiling
%    gamma = the number of inspection intervals necessary to cover x/r_1
%    Param.Q       = infinitesimal generator matrix
%    Param.r       = vector of wear rates
%    Param.f       = LST of the c.d.f. for the shock-damage magnitudes
%    Param.lambda  = shock arrival rate
%    Param.lim     = critical damage threshold
%    Param.budget  = fixed budget for long-run cost
%    Param.cD      = cost of downtime
%    Param.cI      = cost of each inspection
%    Param.cR      = cost of replacement
%    Param.L       = x/r_1
%    Param.states  = dimension of Param.Q
%    Param.I       = identity matrix of dimension Param.states
%    Param.e1      = vector of elements (ith column of Param.I)
%    P_hat         = temp matrix used to calculate stationary probabilities
%    {i,k,n}       = index variables
%    S             = temp variable to store a summed sequence
%************************************************************************
% Load Parameter Information
% ----------------------------------------------------------------------
Param = getappdata(0,'PARAM');
disp(['Function call with \tau =' num2str(tau)]);
X  = []; P  = [];
% ----------------------------------------------------------------------
% The code below is used to obtain the transition probability matrix P.
l = Param.states; % used to keep things simple
gamma = ceil(Param.lim/(min(Param.r)*tau));

```

```
% this populates the transition probability matrix for the process that
% describes the state of the environment at each replacement
for i=1:l
   z0=zeros(size(Param.Q(1,:)));
   z0(i) = 1;
    for k=1:l
        for n = 1:gamma
           Pt = expm(Param.Q*n*tau);
           X(n)   = invt('getG',z0,n*tau);
           PV(n)  = Pt(i,k);
           if (Param.Method(1).Use == 1)
                if ((n > 1) && ((X(n)-X(n-1)) <= Param.Method(2).Param))
                    break;
                end
           end
        end
        P(i,k) = sum((X(2:end)-X(1:(end-1)))*(PV(2:end)'))+ PV(1)*X(1);
    end
end

% ------------------------------------------------------------------------
% Now compute the stationary distribution, p, of P
% ------------------------------------------------------------------------
P_hat      = P - Param.I;
P_hat(:,l) = 1;
p          = Param.p/P_hat;
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% This routine is used to obtain ER_1
% ------------------------------------------------------------------------
ER_1 = zeros(1,l);
for i=1:l
     for n=1:(gamma-1)
        S(n)=invt('getG',Param.I(i,:),n*tau);
     end
     ER_1(i)= tau*(gamma - sum(S));
end
% ------------------------------------------------------------------------

% ------------------------------------------------------------------------
% Compute the limiting average availability
% ------------------------------------------------------------------------
```

```
fx = -(p*Param.EF_1')/(p*ER_1');
% ------------------------------------------------------------------------
% Compute the final costs
% ------------------------------------------------------------------------
cterm = Param.cD*(ER_1 - Param.EF_1) ...
      + Param.cI*floor(ER_1/tau) ...
      + Param.cR*ones(1,l);s
cx = (p*cterm')/(p*ER_1') - Param.budget;
return;
% ------------------------------------------------------------------------
% uniformization
% ------------------------------------------------------------------------
function out = unif(tau,lambda,P,n,numStates)
   pi = 0;
   y  = 1;
   for k = 1:n
        y = y*P*(lambda*tau)/k;
        pi = pi + y;
   end
   out = exp(-lambda*tau)*pi;
return;

function [intLoopCnt, lambda, P] = CalcLoopCnt(tau, Q, epsilon, M)
    K = 0; zi = 1; sigma = 1;
    lambda = max(-diag(Q));
    eta = (1-epsilon)/(exp(-lambda*tau));
    while (sigma <= eta)
        K = K + 1;
        zi = zi*(lambda*tau)/K;
        sigma = sigma + zi;
    end
    P = (lambda^(-1))*(lambda*eye(size(Q))+Q);
    intLoopCnt = K;
return;

function E = expmdemo3(A)
%EXPMDEMO3  Matrix exponential via eigenvalues and eigenvectors.
%   E = EXPMDEMO3(A) illustrates one possible way to compute the matrix
%   exponential.  As a practical numerical method, the accuracy
%   is determined by the condition of the eigenvector matrix.
%
%   See also EXPM, EXPMDEMO1, EXPMDEMO2.

%   Copyright 1984-2004 The MathWorks, Inc.
%   $Revision: 1.1.6.3 $  $Date: 2004/08/16 01:37:28 $

[V,D] = eig(A);
E = V * diag(exp(diag(D))) / V;

return;

function E = expmdemo2(A)
%EXPMDEMO2  Matrix exponential via Taylor series.
%   E = expmdemo2(A) illustrates the classic definition for the
%   matrix exponential.  As a practical numerical method,
%   this is often slow and inaccurate.
%
%   See also EXPM, EXPMDEMO1, EXPMDEMO3.

%   Copyright 1984-2003 The MathWorks, Inc.
%   $Revision: 1.1.6.2 $  $Date: 2004/04/10 23:24:39 $

E = zeros(size(A));
F = eye(size(A));
k = 1;
while norm(E+F-E,1) > 0
   E = E + F;
   F = A*F/k;
   k = k+1;
end

return;

function E = expmdemo1(A)
%EXPMDEMO1  Matrix exponential via Pade approximation.
%   E = EXPMDEMO1(A) is an M-file implementation of the built-in
%   algorithm used by MATLAB for the matrix exponential.
%   See Golub and Van Loan, Matrix Computations, Algorithm 11.3-1.
%
%   See also EXPM, EXPMDEMO2, EXPMDEMO3.

%   Copyright 1984-2003 The MathWorks, Inc.
%   $Revision: 1.1.6.2 $  $Date: 2004/04/10 23:24:38 $

% Scale A by power of 2 so that its norm is < 1/2 .
[f,e] = log2(norm(A,'inf'));
```

```
s = max(0,e+1);
A = A/2^s;

% Pade approximation for exp(A)
X = A;
c = 1/2;
E = eye(size(A)) + c*A;
D = eye(size(A)) - c*A;
q = 6;
p = 1;
for k = 2:q
   c = c * (q-k+1) / (k*(2*q-k+1));
   X = A*X;
   cX = c*X;
   E = E + cX;
   if p
     D = D + cX;
   else
     D = D - cX;
   end
   p = ~p;
end
E = D\E;

% Undo scaling by repeated squaring
for k=1:s, E = E*E; end

return;
```

Appendix B. Availability Parameters File

```
function Param = availability_Param
load RunData;
myCase = RunData.intState;
Param.Method = RunData.Method;

switch myCase
    case {2}
        % 2 state case
        Param.Q       = [-25/3 25/3;25/3 -25/3];
        Param.r       = [13/12 1/4];
        Param.f       = inline('[4.0/(4.0+s) 4.0/(4.0+s)]','s');
        Param.lambda  = 0.50;
        Param.lim     = 1.0;
        Param.budget  = 35;
        Param.cD      = 0.5;
        Param.cI      = 1;
        Param.cR      = 5;
    case {5}
        % 5 state case
        Param.Q       = [-0.5   0.125   0.125   0.125   0.125;
                          0.4    -2  0.4 0.6 0.6;
                          0.025  0.025   -0.1    0.025   0.025;
                          0.05   0.05    0.05    -0.2    0.05;
                          1.5    1   1   1.5 -5];
        Param.r       = [1 2 3 4 10];
        Param.f       = inline('[1 1 1 1 1]*(0.20/(0.20+s))^8','s');
        Param.lambda  = 0.25;
        Param.lim     = 100.0;
        Param.budget  = 0.7;
        Param.cD      = 0.5;
        Param.cI      = 1;
        Param.cR      = 5;
    case {7}
        % exponential with 7 states
        Param.Q     = [-2.8452    0.5466    0.8801    0.1365    0.4692    0.4329    0.3798
                        0.7271   -1.9859    0.1730    0.0118    0.0648    0.2259    0.7833
                        0.3093    0.6946   -4.1467    0.8939    0.9883    0.5798    0.6808
                        0.8385    0.6213    0.2714   -3.5355    0.5828    0.7604    0.4611
                        0.5681    0.7948    0.2523    0.2987   -3.0116    0.5298    0.5678
                        0.3704    0.9568    0.8757    0.6614    0.5155   -4.1742    0.7942
                        0.7027    0.5226    0.7373    0.2844    0.3340    0.2091   -2.7901];
        Param.r = [1 2 3 4 5 6 7]+1.5;
        % uniform 0,10
        Param.f = inline('[1 1 1 1 1 1]*((exp(-s*10)-exp(-s*0))/(s*(10-0)))','s');
        Param.lambda  = 1;
        Param.lim     = 50;
        Param.budget  = 3.5;
        Param.cD      = 0.5;
        Param.cI      = 1;
        Param.cR      = 5;
    case {10}
        % weibull with 10 states
        Param.Q = [...
-3.861608355    0.058187848  0.494874756  0.508179212   ...
 0.065313688    0.486398149  0.41364986   0.933229859   0.374292584 0.527482399
 0.391617223   -3.367934425  0.038333157  0.452239667   ...
 0.375144658    0.496060749  0.560410449  0.259379974   0.249102724 0.545645825
 0.252783651    0.455725962 -3.93287784   0.325584439   ...
 0.37352297     0.843194083  0.268677291  0.204171395   0.924875292 0.284342757
 0.354381927    0.863086892  0.327882957 -4.669336947   ...
 0.484022397    0.806198215  0.784254164  0.049208442   0.629499293 0.370802659
 0.742977968    0.85519697   0.899468788  0.886479964   ...
-6.178942389    0.857785596  0.387870785  0.60616094    0.878308777 0.064692602
 0.650832022    0.472255683  0.313730488  0.76126076    ...
 0.342061094   -4.303955904  0.030983621  0.54634874    0.641674431 0.544809064
 0.939793041    0.78692439   0.251676012  0.883766335   ...
 0.252689256    0.565730368 -5.411183076  0.095837436   0.798390636 0.836375603
 0.832799125    0.655982138  0.432989132  0.457406256   ...
 0.584886917    0.611898522  0.558558528 -4.714868647   0.435026039 0.145321992
 0.469977867    3.98896E-05  0.842382237  0.799202291   ...
 0.523703594    0.102976522  0.200695572  0.442948328 -3.553446558  0.171520257
 0.629865607    0.131237162  0.184488992  0.13407712    ...
 0.163419046    0.158315917  0.087421885  0.066381959  0.095957798 -1.651165486];
        Param.r = [0.4 0.9031 1.4314 1.8062 2.0969 2.3345 2.5353 2.7093 2.8627 3.0000];
        stri = ones(1,10);
        % mean = 2
        % var = 1
        % normally distributed
        Param.f       = inline(['[' int2str(stri) ']*(0.5/(0.5+s))^4'],'s');
        Param.lambda  = 1;
        Param.lim     = 60;
        Param.budget  = 2;
        Param.cD      = 0.5;
        Param.cI      = 1;
        Param.cR      = 5;
    case {20}
        Param.Q = [...
-10.068   0.068   0.599   0.302   0.562   0.018   0.813   0.369   0.519   0.953   0.425   0.681 ...
  0.685   0.556   0.744   0.790   0.340   0.514   0.833   0.295;
  0.722  -9.370   0.336   0.460   0.701   0.404   0.695   0.716   0.141   0.624   0.058   0.856 ...
  0.974   0.689   0.489   0.331   0.019   0.046   0.476   0.632;
  0.841   0.453 -10.381   0.503   0.285   0.800   0.441   0.286   0.316   0.458   0.797   0.097 ...
  0.280   0.761   0.650   0.782   0.768   0.991   0.340   0.531;
  0.821   0.211   0.118 -10.014   0.987   0.160   0.471   0.789   0.861   0.648   0.560   0.392 ...
  0.174   0.824   0.042   0.777   0.458   0.310   0.542   0.871;
  0.544   0.735   0.520   0.944  -10.037  0.312   0.881   0.205   0.433   0.979   0.771   0.376 ...
  0.922   0.528   0.335   0.105   0.044   0.256   0.562   0.584;
  0.364   0.577   0.995   0.650   0.501  -9.756   0.645   0.430   0.059   0.421   0.995   0.516 ...
  0.260   0.342   0.433   0.329   0.100   0.610   0.925   0.603;
  0.501   0.789   0.313   0.946   0.430   0.591 -10.015   0.629   0.886   0.976   0.090   0.894 ...
  0.375   0.068   0.099   0.261   0.957   0.447   0.274   0.488;
  0.330   0.083   0.506   0.464   0.516   0.054   0.212  -8.547   0.256   0.519   0.721   0.642 ...
  0.645   0.264   0.681   0.388   0.609   0.418   0.296   0.943;
  0.231   0.376   0.820   0.271   0.047   0.869   0.126   0.246  -8.582   0.543   0.997   0.203 ...
  0.316   0.994   0.027   0.897   0.247   0.412   0.613   0.347;
  0.605   0.201   0.299   0.069   0.965   0.728   0.536   0.657   0.224  -8.860   0.108   0.069 ...
  0.192   0.385   0.321   0.767   0.930   0.714   0.891   0.200;
  0.047   0.646   0.709   0.187   0.066   0.017   0.840   0.132   0.591   0.849  -7.546   0.800 ...
  0.233   0.786   0.052   0.244   0.430   0.103   0.714   0.098;
  0.737   0.124   0.869   0.349   0.510   0.744   0.240   0.513   0.564   0.546   0.133  -8.526 ...
  0.220   0.870   0.370   0.455   0.049   0.380   0.202   0.652;
  0.488   0.508   0.656   0.914   0.705   0.292   0.022   0.274   0.866   0.815   0.704   0.892 ...
-10.250   0.085   0.384   0.272   0.905   0.752   0.108   0.606;
  0.986   0.106   0.862   0.014   0.625   0.756   0.148   0.587   0.220   0.486   0.306   0.992 ...
  0.124  -9.647   0.336   0.423   0.978   0.149   0.738   0.811;
  0.978   0.029   0.927   0.988   0.227   0.104   0.476   0.268   0.301   0.116   0.064   0.401 ...
  0.538   0.845  -9.503   0.699   0.658   0.661   0.762   0.462;
  0.142   0.051   0.939   0.174   0.631   0.227   0.878   0.784   0.587   0.143   0.174   0.341 ...
  0.689   0.466   0.238  -7.801   0.727   0.237   0.279   0.094;
  0.929   0.183   0.666   0.403   0.153   0.488   0.863   0.149   0.902   0.785   0.193   0.317 ...
  0.720   0.599   0.081   0.825  -9.931   0.728   0.845   0.103;
  0.016   0.510   0.598   0.399   0.311   0.873   0.996   0.773   0.524   0.269   0.670   0.364 ...
  0.939   0.208   0.301   0.403   0.686  -10.340  0.624   0.875;
  0.728   0.648   0.405   0.786   0.649   0.579   0.757   0.520   0.669   0.557   0.387   0.616 ...
  0.008   0.517   0.180   0.652   0.363   0.470  -9.550   0.061;
  0.734   0.510   0.134   0.506   0.836   0.421   0.886   0.587   0.677   0.136   0.070   0.597 ...
  0.582   0.158   0.716   0.152   0.401   0.858   0.955  -9.915;];
```

```
        r = [0.4444 1.7778 4.0000 7.1111 11.1111 16.0000 21.7778 28.4444];
        r = [r 36.0000 44.4444 53.7778 64.0000 75.1111 87.1111 100.0000];
        Param.r = [r 113.7778 128.4444 144.0000 160.4444 177.7778];
        Param.r = Param.r + 0.2;
        % gamma distributed shock distribution
        %theta = 1;
        %k = 7
        stri          = ones(1,20);
        Param.f       = inline(['[' int2str(stri) ']*(1+2*s)^4'],'s');
        Param.lambda  = 1;
        Param.lim     = 100;
        Param.budget  = 1.2;
        Param.cD      = 0.5;
        Param.cI      = 1;
        Param.cR      = 5;
    otherwise
        erstr = '%g is not a specified state.';
        error(erstr,myCase)
        r = 0;
end
Param.L      = Param.lim/min(Param.r);
Param.states = length(Param.r);
Param.R_D    = diag(Param.r);
Param.I      = eye(Param.states);
Param.e1     = ones(Param.states,1);  % Create column vector of ones
% now perform one set of calculations for the invt function
m=11; c=[]; ga=8; A=ga*log(10);
Param.mm=2^m;
Param.c = [1 11 55 165 330 462 462 330 165 55 11 1];
Param.u = exp(A/2)/Param.lim;
Param.x = A/(2*Param.lim);
Param.h = pi/Param.lim;
Param.su= zeros(m+2);
% calculate the p-values
Param.p                     = zeros(1,Param.states);
Param.p(1,Param.states)     = 1;
% calculate the numerator
EF_1       = zeros(1,Param.states);
for i = 1:Param.states
    EF_1(i) = invt('getE',Param.I(i,:),1.0, Param);
end
Param.EF_1 = EF_1;
return
```

Appendix C. Laplace Inversion Code

```
%*************************************************************************
% invt:  accomplishes numerical inversion of laplace transforms
% -----------------------------------------------------------------------
% Called by: IFS
% Calls:     feval, dot, zeros
% VARIABLES:
%  EG       = full array (n x 1)
%  z0       = individual item to trim (scalar)
%  ntau     = trimmed array
%  c        = index of individual item to trim (scalar)
%  u        = length of the full item to trim
%  rho,qx,m,c,ga,mm,ntr,u,x,h,su,sm,f1 = algorithmic parameters
%  k        = index variable
%  j        = sqrt(-1)
%******************************************************************************
% References:
%            Abate, J. and W. Whitt (1995).  Numerical inversion of the
%            Laplace transform of probability distributions. ORSA
%            Journal on Computing, 7, 36-43.
function f1 = invt(EG,z0,ntau, Param)
if (nargin < 4)
    Param = getappdata(0,'PARAM');
end
ntr=15;
sm= feval(EG,Param.x,z0,ntau,Param.f(Param.x),Param)/2;
for k=1:ntr
    sm = sm + ((-1)^k)*...
        feval(EG,Param.x+k*Param.h*j,z0,ntau,Param.f(Param.x+k*Param.h*j),Param);
end
Param.su(1)=sm;
for k=1:12
    n = ntr+k;
    Param.su(k+1) = Param.su(k) + ((-1)^n)*...
        feval(EG,Param.x+n*Param.h*j,z0,ntau,Param.f(Param.x+n*Param.h*j),Param);
end
f1 = Param.u*(dot(Param.c,Param.su(1:12,1)'))/Param.mm;
return;
% ----------------------------------------------------------------------
% getE
% ----------------------------------------------------------------------
function out_getE = getE(s,z0,ntau,f_eval, Param)
```

```
%     R_D = diag(r);
F     = diag(f_eval);
%     I   = eye(size(Q));
%     e1  = ones(size(Q,1),1);  % Create column vector of ones
if (nargin < 4)
    Param = getappdata(0,'PARAM');
end
% Compute the conditional expectation values of F_1.
z = (1/s)*(z0*inv(s*Param.R_D-Param.Q-Param.lambda*(F-Param.I)))*Param.e1*ntau;

% The desired value is the real part of z.
out_getE = real(z);
return;
% ------------------------------------------------------------------------
% getG
% ------------------------------------------------------------------------
function out_getG = getG(s,z0,ntau,f_eval, Param)
F     = diag(f_eval);
if (nargin < 5)
    Param = getappdata(0,'PARAM');
end
A2 = expm((Param.Q+Param.lambda*(F-Param.I)-s*Param.R_D)*ntau);

% Compute the cdf values for given t (ntau).
z  = (1/s)*(1-z0*A2*Param.e1); % get the cdf

% The desired value is the real part of z.
out_getG = real(z);
return;
```

Appendix D. MADS Batch Execution

```
function [BestF,BestI,RunStats,RunSet] = mads_batch
%MADS_BATCH  Sets up and runs the MADS algorithm without a GUI.
%
%   Syntax:
%      mads_batch
%
%   Description:
%      This function serves as a GUI-free alternative to NOMADm in setting
%      up an optimization problem, setting various algorithm parameters and
%      user options, and calling the MADS optimizer.  It first sets all of the
%      variables to their default values, which are clearly stated in the
%      MADS_DEFAULTS file.  To change a variable from its default value, the
%      user must add a statement to this file to do so.  Some variable
%      statements are included here for convenience, which can be change
%      manually.
%
%   See also MADS_DEFAULTS, MADS

%*****************************************************************************
%   Copyright (c) 2001-2005 by Mark A. Abramson
%
%   This file is part of the NOMADm software package.
%
%   NOMADm is free software; you can redistribute it and/or modify it under the
%   terms of the GNU General Public License as published by the Free Software
%   Foundation; either version 2 of the License, or (at your option) any later
%   version.
%
%   NOMADm is distributed in the hope that it will be useful, but WITHOUT ANY
%   WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
%   FOR A PARTICULAR PURPOSE.  See the GNU General Public License for more
%   details.
%
%   You should have received a copy of the GNU General Public License along
%   with NOMADm; if not, write to the Free Software Foundation, Inc.,
%   59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
% -----------------------------------------------------------------------------
%   Originally created, 2001.
%   Last modified, 1 March 2006 by Tim Booher
%
%   Author information:
```

```
%   Mark A. Abramson, LtCol, USAF, PhD
%   Air Force Institute of Technology
%   Department of Mathematics and Statistics
%   2950 Hobson Way
%   Wright-Patterson AFB, OH 45433
%   (937) 255-3636 x4524
%   Mark.Abramson@afit.edu
%*******************************************************************************

%*******************************************************************************
% Calls:     mads_defaults, mads, < user initial points file >
% Variables:
%  Defaults    = structure of MADS default values (see mads_defaults)
%  Options     = structure for options settings (see mads_defaults)
%  problemPath = location of user problem files
%  Problem     = structure of data for optimization problem
%  newPath     = logical indicating if path is not the Matlab path
%  iterate0    = structure of data for the initial iterate (see mads)
%  BestF       = final best feasible solution found
%  BestI       = final least infeasible solution found
%  RunStats    = structure of MADS Run statistics (see mads)
%*******************************************************************************

% Set Options to their default values
clear variables
Defaults = mads_defaults('Truth');
Options  = Defaults.Options;
Problem.nameCache = 'CACHE';

% Specify Problem Files
problemPath  = pwd;
Problem.File.F = 'availabilityCalc';             % functions file
Problem.File.O = 'availabilityCalc_Omega';       % linear constraints file
Problem.File.X = 'availabilityCalc_X';           % closed constraints file
Problem.File.I = 'availabilityCalc_x0';          % initial points file
Problem.File.N = 'availabilityCalc_N';           % discrete neighbor file (MVP only)
Problem.File.P = 'availabilityCalc_Param';       % parameter file
Problem.File.C = 'availabilityCalc_Cache.mat';   % previously created Cache file
Problem.File.S = 'availabilityCalc_Session.mat'; % previously created Session file
Problem.File.H = 'availabilityCalc_History.txt'; % iteration history text file
Problem.fType  = 'M';                     % type of functions file {M,F,C}
Problem.nc     = 1;                       % number of nonlinear constraints

```

```
% Set the path, and load any user-provided problem parameters
cwd = pwd; cd(problemPath);
if (exist(Problem.File.P,'file') == 2)
   Problem.Param = feval(Problem.File.P);
   setappdata(0,'PARAM',Problem.Param);
end

% Specify Choices for SEARCH
Options.Search(1).type    = 'None';      % For choices, see mads_defaults
Options.Search(1).nIter   = 0;           % Number of iterations for Search #1
Options.Search(1).nPoints = 0;           % Number of poll or sample points
Options.Search(1).file    = '';          % filename must include the full path
Options.Search(1).local   = 0;           % flag to turn on trust region
Options.Search(1).merit   = 0;           % flag to penalize clustered data
Options.Search(2).type    = 'None';      % For choices, see mads_defaults
Options.Search(2).nIter   = Inf;         % Number of iterations for Search #2
Options.Search(2).nPoints = 0;           % Number of poll or sample points
Options.Search(2).file    = '';          % filename must include the full path
Options.Search(2).local   = 0;           % flag to turn on trust region
Options.Search(2).merit   = 0;           % flag to penalize clustered data
Options.nSearches         = 2;
Options.SurOptimizer      = 'fmincon';

% Specify Choices for POLL
Options.pollStrategy    = 'Standard_2n'; % For choices, see mads_defaults
Options.pollOrder       = 'Consecutive'; % For choices, see mads_defaults
Options.pollCenter      = 0;             % Poll around n-th filter point
Options.pollComplete    = 0;             % Flag for complete polling

% Specify Termination Criteria
Options.Term.delta      = 1e-4;          % minimum mesh size
Options.Term.nIter      = Inf;           % maximum number of iterations
Options.Term.nFunc      = 50000;         % maximum number of function evals
Options.Term.time       = Inf;           % maximum CPU time
Options.Term.nFails     = Inf;           % max number of consecutive Poll fails

% Choices for Mesh Control
Options.delta0          = 1;             %Problem.Param.L/4;     % initial mesh size
Options.deltaMax        = Inf;           % bound on how coarse the mesh can get
Options.meshRefine      = 0.5;           % mesh refinement factor
Options.meshCoarsen     = 1.0;           % mesh coarsening factor

% Choices for Filter management (for problems with nonlinear constraints)
```

```
Options.hmin            = 1e-4;          % minimum infeasible point h-value
Options.hmax            = 1.0;           % maximum h-value of a filter point

% Choices for EXTENDED POLL (for MVP problems)
Options.ePollTriggerF   = 0.01;          % f-value Extended Poll trigger
Options.ePollTriggerH   = 0.01;          % h-value Extended Poll trigger

% MADS flag parameter values
Options.loadCache        = 1;            % load pre-existing Cache file
Options.countCache       = 1;            % count Cache points as function calls
Options.runStochastic    = 0;            % runs problem as a stochastic problem
Options.scale            = 2;            % scale directions using this log base
Options.useFilter        = 0;            % filter (0=none, 1=multi-pt, 2=2-pt)
Options.degeneracyScheme = 'random';     % scheme for degenerate constraints
Options.removeRedundancy = 0;            % discard redundant linear constraints
Options.computeGrad      = 0;            % compute gradient, if available
Options.saveHistory      = 0;            % saves MADS performance to text file
Options.plotHistory1     = 0;            % plot MADS performance
Options.plotHistory2     = 0;            % plot MADS performance real-time
Options.plotFilter       = 0;            % plot the filter real-time
Options.plotColor        = 'k';          % color of history plot

% Set up figure handles for real-time plots
if (Problem.nc == 0)
   Options.plotFilter = 0;
end
if (Options.plotFilter)
   figure; Options.fplothandle = gca;
end
if (Options.plotHistory2)
   figure; Options.hplothandle = gca;
end

% Get the initial iterates and call the optimizer
iterate0 = feval(Problem.File.I);
[BestF,BestI,RunStats,RunSet] = mads(Problem,iterate0,Options);

% Perform any user-defined post-processing (must have argument)
if (exist(Problem.File.P,'file') == 2) && (nargin(Problem.File.P) < 0)
   Param = feval(Problem.File.P,BestF); setappdata(0,'PARAM',Param);
end
cd(cwd);
return
```

Appendix E. GPS Constraints File

```
%*********************************************************************
% availability_Omega:  User-supplied function for defining Omega, based on p.
% --------------------------------------------------------------------
%   Variables:
%     A  = Coefficient matrix for bound and linear constraints
%     l  = Lower bounds for A*x for any iterate x
%     u  = Upper bounds for A*x for any iterate x
%*********************************************************************
function [A,l,u] = example_Omega(n);
Param = getappdata(0,'PARAM');
A = eye(n);
l = [0.01];
u = [Param.L];
return;
```

Appendix F. Initial Iterates File

```
function iterate = availabilityCalc_x0;
% iterate.p is a vector containing the values of the continuous variables
% iterate.x is a cell array containing the values of the categorical
Param = getappdata(0,'PARAM');
iterate(1).x = Param.L/2;       % start in the middle of the lifetime
iterate(1).p = {};  % null
return;
```

Appendix G. MADS Batch Script

```
function myoutput = CallMadsBatch(intSt,i,rnum, origDir)
%CallMadsBatch: function to set parameters and call a mads batch run
%
%   Syntax:
%     CallMadsBatch(States, Method, RunID, OriginalDirectory);
%
%   Description:
%   This Matlab program
%
%   Calls: mads_batch, int2str, disp, save, tic, toc
%
%****************************************************************************
%   Copyright (c) 2006 by Timothy B. Booher
% ---------------------------------------------------------------------------
%   Originally created, 2005.
%   Last modified, 14 January 2006
%
%   Author information:
%   Timothy B. Booher, Capt, USAF
%   Air Force Institute of Technology
%   Department of Operational Sciences
%   2950 Hobson Way
%   Wright-Patterson AFB, OH 45433
%   Timothy.Booher@afit.edu
%****************************************************************************
%
% VARIABLES (only for the CallMadsBatch function):
%*INPUT*VARIABLES***********************************************************
%  tau = the inter-inspection interval and decision variable
%  intSt   = the state
%  i       = the method parameter case
%  rnum    = a unique id number to describe the current run
%  origDir = the directory with the location of the original run files
%*FUNCTION*VARIABLES********************************************************
%    RunData.Method(n).Use   = boolean (1,0) that states if the method should
%                              be used in case 'n'
%    RunData.Method(n).Param = parameter for a particular method
%    BestF,BestI,RunStats, RunSet = mads output variables (see mads.m)
%    runTime.time  = my record of the program run time
%**********************************************************************
% set parameter file
```

```
RunData.intState  = intSt;
% Method 1  truncation
% Method 2  Matrix exponential via Uniformization
% Method 3  Matrix exponential via eigenvalues and eigenvectors
% Method 4  Matrix exponential via Taylor series.
% Method 5  Matrix exponential via Pade approximation.
% run1 = baseline | run2 = trunc | run3 = unif | run4 = eig | run5 = Taylor | run6 = Pade
if (nargin > 3)
    path(path,origDir)
end
blnUseFlops = true;
if blnUseFlops
    if (exist('C:\Program Files\ICL\WinPAPI\PAPI MATLAB Support\flops.dll')~=3)
       path('C:\Program Files\ICL\WinPAPI\PAPI MATLAB Support\',path);
       disp('path added for FLOPS');
    else
       disp('path addition not needed -- FLOPS in use');
    end
end
% save structure in .mat file
switch i
    case 1
        % RunData -- contains information on each run
        %         *$*$*$* RUN 1: Baseline *$*$*$*
        % 1:gamma scaling | 2:truncation | 3:second loop tol | 4:uniformization
        % ***** what methods do we want to run?
        RunData.Method(1).Use    = false;
        RunData.Method(2).Use    = false;
        RunData.Method(3).Use    = false;
        RunData.Method(4).Use    = false;
        RunData.Method(5).Use    = false;
        % ***** what parameters needed for each method?
        RunData.Method(1).Param  = 0;
        RunData.Method(2).Param  = 0;
        RunData.Method(3).Param  = 0;
        RunData.Method(4).Param  = 0;
        RunData.Method(4).Param  = 0;
    case 2
        %         *$*$*$* RUN 2: Truncation *$*$*$*
        % ***** what methods do we want to run?
        RunData.Method(1).Use    = true;
        RunData.Method(2).Use    = false;
        RunData.Method(3).Use    = false;
        RunData.Method(4).Use    = false;
        RunData.Method(5).Use    = false;
        % ***** what parameters needed for each method?
        RunData.Method(1).Param  = 1e-4;
        RunData.Method(2).Param  = 0;
        RunData.Method(3).Param  = 0;
        RunData.Method(4).Param  = 0;
        RunData.Method(5).Param  = 0;
    case 3
        %         *$*$*$* RUN 3: Uniformization *$*$*$*
        % 1:gamma scaling | 2:truncation | 3:second loop tol | 4:uniformization
        % ***** what methods do we want to run?
        RunData.Method(1).Use    = false;
        RunData.Method(2).Use    = true;
        RunData.Method(3).Use    = false;
        RunData.Method(4).Use    = false;
        RunData.Method(5).Use    = false;
        % ***** what parameters needed for each method?
        RunData.Method(1).Param  = 0;
        RunData.Method(2).Param  = 1e-2;
        RunData.Method(3).Param  = 0;
        RunData.Method(4).Param  = 0;
        RunData.Method(5).Param  = 0;
    case 4
        %         *$*$*$* RUN 4: expm via eigenvalues and eigenvectors *$*$*$*
        % 1:gamma scaling | 2:truncation | 3:second loop tol | 4:uniformization
        % ***** what methods do we want to run?
        RunData.Method(1).Use    = false;
        RunData.Method(2).Use    = false;
        RunData.Method(3).Use    = true;
        RunData.Method(4).Use    = false;
        RunData.Method(5).Use    = false;
        % ***** what parameters needed for each method?
        RunData.Method(1).Param  = 0;
        RunData.Method(2).Param  = 0;
        RunData.Method(3).Param  = 0;
        RunData.Method(4).Param  = 0;
        RunData.Method(5).Param  = 0;
    case 5
        %         *$*$*$* RUN 4: Matrix exponential via Taylor series. *$*$*$*
        % 1:gamma scaling | 2:truncation | 3:second loop tol | 4:uniformization
        % ***** what methods do we want to run?
        RunData.Method(1).Use    = false;
        RunData.Method(2).Use    = false;
        RunData.Method(3).Use    = false;
        RunData.Method(4).Use    = true;
        RunData.Method(5).Use    = false;
        % ***** what parameters needed for each method?
        RunData.Method(1).Param  = 0;
        RunData.Method(2).Param  = 0;
        RunData.Method(3).Param  = 0;
        RunData.Method(4).Param  = 0;
        RunData.Method(5).Param  = 0;
    otherwise
        %         *$*$*$* RUN 6: expm via Pade approximation *$*$*$*
        % 1:gamma scaling | 2:truncation | 3:second loop tol | 4:uniformization
        % ***** what methods do we want to run?
        RunData.Method(1).Use    = false;
        RunData.Method(2).Use    = false;
        RunData.Method(3).Use    = false;
        RunData.Method(4).Use    = false;
        RunData.Method(5).Use    = true;
        % ***** what parameters needed for each method?
        RunData.Method(1).Param  = 0;
        RunData.Method(2).Param  = 0;
        RunData.Method(3).Param  = 0;
        RunData.Method(4).Param  = 0;
        RunData.Method(5).Param  = 0;
end
disp(['Starting batch run for state ' int2str(intSt)]);
save RunData;

% call the run
tic;
    [BestF,BestI,RunStats,RunSet] = mads_batch;
myoutput = toc;

% output results to a text file
intN  = [int2str(intSt) int2str(i) int2str(rnum)];
strFN = ['mads_output' intN];
% output results to a .mat file
save(strFN);

myoutput = BestF;

return;
```

Appendix H. Post-processing Code

```
% Tim Booher
% 2 March 2006 -- This script postprocesses batch MADS data files
%
% First, data are conditioned, then placed in an excel file
clear all;
strRpt          = 'results_final.xls';
intMethods      = 1:6;
runCount        = 30;
states          = [2 5 7 10 20];
stateSize       = length(states);
workdir         = pwd;
lm              = length(intMethods);

TAU   = zeros(stateSize,lm);
A     = zeros(stateSize,lm);
FCALL = A;
C     = A;
RTIME = A;
RUNTIMES = zeros(1,lm);
% collect the data
for intCurSt = states
    k = find(states==intCurSt);
    for myrun = 1:runCount
      mr = int2str(myrun);
        for j = intMethods
            CurCol = find(intMethods==j);
            cs = int2str(intCurSt); % current state
            cm = int2str(j);   % current method
            mydir = [cs cm mr];
            fname = ['mads_output' cs cm mr '.mat'];
            disp(fname);
            if (exist(fname)==2)
                load(fname);
                STATE(k).TAU(myrun,CurCol)   = BestF.x;
                STATE(k).A(myrun,CurCol)     = -BestF.f;
                STATE(k).C(myrun,CurCol)     = -BestF.c;
                STATE(k).FCALL(myrun,CurCol) = RunStats.nFunc;
                STATE(k).RTIME(myrun,CurCol) = RunStats.time;
            else
                STATE(k).TAU(myrun,CurCol)   = NaN;
                STATE(k).A(myrun,CurCol)     = NaN;
                STATE(k).C(myrun,CurCol)     = NaN;
                STATE(k).FCALL(myrun,CurCol) = NaN;
                STATE(k).RTIME(myrun,CurCol) = NaN;
            end
        end
    end
    SUMT(k).TAU.HW     = 1.96*(sqrt(var(STATE(k).TAU))/sqrt(runCount));
    SUMT(k).TAU.Mean   = mean(STATE(k).TAU);
    SUMT(k).A.HW       = 1.96*(sqrt(var(STATE(k).A))/sqrt(runCount));
    SUMT(k).A.Mean     = mean(STATE(k).A);
    SUMT(k).C.HW       = 1.96*(sqrt(var(STATE(k).C))/sqrt(runCount));
    SUMT(k).C.Mean     = mean(STATE(k).C);
    SUMT(k).FCALL      = mean(STATE(k).FCALL);
    SUMT(k).RTIME.HW   = 1.96*(sqrt(var(STATE(k).RTIME))/sqrt(runCount));
    SUMT(k).RTIME.Mean = mean(STATE(k).RTIME);
    SUMT(k).TVAR       = var(STATE(k).RTIME);
    RUNTIMES = [RUNTIMES; STATE(k).RTIME];
end

% output the data
AVGM = zeros(5,stateSize*lm);
RAW  = zeros(runCount*5,stateSize*lm);
EXPM = zeros(stateSize*2,length(intMethods));
myCol = 0; expmrow = -1;
for j = states
    k = find(states==j);
    expmrow = expmrow + 2;
    myExpmCol = 0;
    for i = intMethods
        curCol = find(intMethods==i);
        myCol = myCol + 3;
        myExpmCol = myExpmCol + 1;
        AVGM(1,myCol-2) = SUMT(k).TAU.Mean(curCol)-SUMT(k).TAU.HW(curCol);
        AVGM(1,myCol-1) = SUMT(k).TAU.Mean(curCol);
        AVGM(1,myCol)   = SUMT(k).TAU.Mean(curCol)+SUMT(k).TAU.HW(curCol);
        AVGM(2,myCol-2) = SUMT(k).A.Mean(curCol)-SUMT(k).A.HW(curCol);
        AVGM(2,myCol-1) = SUMT(k).A.Mean(curCol);
        AVGM(2,myCol)   = SUMT(k).A.Mean(curCol)+SUMT(k).A.HW(curCol);
        AVGM(4,myCol-2) = SUMT(k).FCALL(curCol);
        AVGM(4,myCol-1) = NaN;
        AVGM(4,myCol)   = NaN;
        AVGM(3,myCol-2) = SUMT(k).RTIME.Mean(curCol)-SUMT(k).RTIME.HW(curCol);
        AVGM(3,myCol-1) = SUMT(k).RTIME.Mean(curCol);
        AVGM(3,myCol)   = SUMT(k).RTIME.Mean(curCol)+SUMT(k).RTIME.HW(curCol);
        AVGM(5,myCol-2) = SUMT(k).TVAR(curCol);
        AVGM(5,myCol-1) = NaN;
        AVGM(5,myCol)   = NaN;
        EXPM(expmrow, myExpmCol)   = SUMT(k).RTIME.Mean(curCol);
        EXPM(expmrow+1, myExpmCol) = sqrt(SUMT(k).TVAR(curCol));
        stRow = 1;
        for rc = 1:runCount
            RAW(stRow,myCol)   = STATE(k).TAU(rc,curCol);
            RAW(stRow+1,myCol) = STATE(k).A(rc,curCol);
            RAW(stRow+2,myCol) = STATE(k).FCALL(rc,curCol);
            RAW(stRow+3,myCol) = STATE(k).RTIME(rc,curCol);
            RAW(stRow+4,myCol) = NaN;
            stRow = stRow + 5;
        end
    end
end

xlswrite(strRpt, AVGM, 'summary', 'B3');
xlswrite(strRpt, RAW, 'raw', 'B3');
xlswrite(strRpt, EXPM, 'expm', 'B3');
xlswrite(strRpt, RUNTIMES, 'runtimes', 'B3');

disp('done');
```

Appendix I. Plot Generator

```
function [A,cr,cd,ci,ERt,mym,tau] = Controller(intState)
% ***************    Enumeration Tau Plotter *****************************
% Tim Booher -- timothy.booher@afit.edu
% 051018
% ***********************************************************************
%intState      = 2;         % # of states in random environment
run_num       = intState;   % unique id to save files associated with this run
blnUseFlops   = false;
blnOutputText = false;      % create text file?
blnProdPlots  = false;      % produce plots
MOC           = 2;          % which method are we concerned with? (in plots)
inc           = 25;         % how many increments?
mult          = 0.3;        % how far from \tau = 0?
sp            = 1;          % what range of the overall \Lambda do we cover?
% RunInfo -- contains information on each run
% *******************************************************************
%                              *$*$*$* RUN 1 *$*$*$*
% ***** name for the run
RunInfo(1).runTitle         = [num2str(intState) ' state with uniformization'];
% ***** numerical id for the run
RunInfo(1).runID            = 1;
% 1:gamma scaling | 2:first loop tol | 3:second loop tol | 4:uniformization
% ***** what methods do we want to run?
RunInfo(1).Method(1).Use    = true;
RunInfo(1).Method(2).Use    = false;
RunInfo(1).Method(3).Use    = false;
RunInfo(1).Method(4).Use    = false;
% ***** what parameters needed for each method?
RunInfo(1).Method(1).Param  = 1e-3;
RunInfo(1).Method(2).Param  = 1e-3;
RunInfo(1).Method(3).Param  = 0;
RunInfo(1).Method(4).Param  = 100;

% ***********************************************************************
intIter = 0;

if blnUseFlops
    if (exist('C:\Program Files\ICL\WinPAPI\PAPI MATLAB Support\flops.dll')~=3)
        path('C:\Program Files\ICL\WinPAPI\PAPI MATLAB Support\',path);
        disp('path added for FLOPS');
    else
```

```
        disp('path addition not needed -- FLOPS in use');
    end
end
% **********************************************************************
% create output file
if (blnOutputText)
    fid = fopen(['output' int2str(run_num) '.csv'],'wt');
    fid2 = fopen(['watch' int2str(run_num) '.csv'],'wt');
    fid3 = fopen(['p_rpt' int2str(run_num) '.csv'],'wt');
end
% **********************************************************************
% initiate loop over all methods
for i = 1:length(RunInfo)
    if (blnUseFlops)
        flops(0);
    else
        tic;
    end
    % ******************************************************************
    % save RunInfo into .mat file
    RunInfo(i).intState = intState;
    RunData = RunInfo(i);
    save RunData;
    % ******************************************************************
    % get inputs
    Param = feval('availabilityCalc_Param');
    setappdata(0,'PARAM',Param);
    Lambda = sp*Param.lim/min(Param.r);
    % ******************************************************************
    % build tau vector
    tau = linspace(mult*Lambda/inc,Lambda-Lambda/(inc*mult),inc);
    %lt = length(tau); % i think this can be 'inc'
    A = []; cr = []; cd = []; ci = []; ERt = [];
    % ******************************************************************
    % Perform Iterations over all \tau
    mym = 1:inc;
    for m = 1:inc
        if (Param.lim/(min(Param.r)*tau(m)) <= 1)
            error('\tau is too big relative to \Lambda -- add increments');
            break;
        end
        gamma = ceil(Param.lim/(min(Param.r)*tau(m)));
        fprintf(1,': %s :',RunInfo(i).runTitle);
        fprintf(1,' Iteration:\t %g of %g',m,inc);
        fprintf(1,' \t tau: %g \t gamma: %g\n',tau(m), gamma);
        [A(m), cr(m), cd(m), ci(m), ERt(m)] = availabilityCalc(tau(m));
    end
    % *******************************************************************
    % Output Performance Metrics to Systerm
    if (~blnUseFlops)
        ops(i) = toc;
    else
        [ops(i), mflops(i)] = flops;
    end
    % *******************************************************************
    % calculate performance on any iteration other than 1 (baseline)
    for j = 2:i
        pim = 100*((ops(1)-ops(j))/ops(1));
        mse = mean((AC{j}-AC{1}).^2);
        if (blnOutputText)
            fprintf(fid3,'%g, %g, ', run_num, pim);
            fprintf(fid3,'%g, %g', mse, intIter);
        end
    end
    % *******************************************************************
    % Output Data to text file for post-processing -- used on remote unix
    % system (HPC)
    if (blnOutputText)
        for b = 1:inc
            fprintf(fid,'%g',tau(b));
            fprintf(fid,', %g',A(b));
            fprintf(fid,', %g',C(b));
            intM = [RunInfo(i).Method(1).Use RunInfo(i).Method(2).Use];
            intM = [intM RunInfo(i).Method(3).Use RunInfo(i).Method(4).Use];
            fprintf(fid,', %s',mat2str(intM));
            fprintf(fid,', %g',run_num);
            fprintf(fid,'\n');
        end
        save(['output_' int2str(run_num)], 'A', 'C', 'tau');
    end
end
% ***********************************************************************
% Display final results
% ***********************************************************************
% Produce Plots
if (blnProdPlots)
```

```
    figure; hold on;
    myLineSpec{1} = '-k';
    myLineSpec{2} = '-or';
    myLineSpec{3} = '-+g';
    myLineSpec{4} = '--xc';
    myLineSpec{5} = '--sm';
    myLineSpec{6} = '--db';
    la = length(AC);
    if (la > 6)
        warning('More methods used than linespecs available.');
    end
    for j = 1:la
        plotyy(tau,-AC{j},tau,CC{j});
    end
    ylabel('Availability(\tau)');
    xlabel('\tau');
    %      legend(plts,legStr);
    title([int2str(intState) '-state case with '...
        int2str(inc) ' increments']);
    grid on;
    myx = get(gca,'xlim'); myy = get(gca,'ylim');
    myx = myx(1)+(myx(2)-myx(1))*(1/2);
    myy = myy(1)+(myy(2)-myy(1))*(1/2);
    % *******************************************************************
    % calculate display improvement
    % *******************************************************************
    for j = 2:i
        text(myx,myy,['MSE Method ' num2str(j) ' = ' num2str(mse)],...
            'BackgroundColor','white');
        text(myx,myy,['Improvement Method ' num2str(j) ' = ' num2str(pim)],...
            'BackgroundColor','white');
        text(myx,myy,['Method ' num2str(j) ' Prm = ' ...
            num2str(RunInfo(j).Method(MOC).Param)],'BackgroundColor','white');
    end
end
if (blnOutputText)
    fclose(fid); fclose(fid2); fclose(fid3);
end
return;
```

Appendix J. Perl Batch Controller

```
#!/bin/perl -w
use Cwd; use File::Copy;
my $stdir = getcwd();                  # get current directory
print "starting dir is $stdir\n";
my $nr = 1; my $i = 0;                 # initialize index
foreach $st (qw/2 5 7 10 20/) {        # iterate over all states
    @j = qw/5 10/;               # array of times for each state
    for ($k=1; $k <= 6; $k++) {        # 6 different methods
        for ($r=1; $r <= $nr; $r++) {  # iterate over each run
            $r = sprintf("%02d",$r);   # format the last two digits
            $rnum=$st.$k.$r;           # unique id for each run
            print "submitting $i:$st \t $rnum \t k: ($k) \t $j[$i]\n";
            $myscript = <<"ENDSCRIPT";
#!/bin/sh
#BSUB -q regular
#BSUB -n 1
#BSUB -W $j[$i]:00
#BSUB -a SMP
#BSUB -J b_$rnum
#BSUB -o matlab_run_$rnum.out
#BSUB -e matlab_run_$rnum.out
#BSUB -P WPBAFITO25047ACM
matlab -nojvm -nodesktop -nosplash > matlab_run_$rnum.out << EOL
CallMadsBatch($st,$k,$r,$stdir);
EOL
ENDSCRIPT
        mkdir($rnum,0777);          # create a directory with full permissions
        system "cp *.m $rnum/";
        chdir $rnum;                # place all files in that directory
        $myfilename = "booher_job_$rnum.sh";
        open DUMMYFILE, ">$myfilename"; print DUMMYFILE $myscript;
        close DUMMYFILE;
        system "bsub < $myfilename"; # submit the script to the lsf engine
        chdir $stdir;
        #print $rnum."\n";
        }
    }
    $i+=1;
}
```

Bibliography

1. Abate, J., Choudhury, G., and Whitt, W. (1999). *Computational Probability.* Kluwer, Boston.

2. Abate, J. and Whitt, W. (1992). The Fourier-series method for inverting transforms of probability distributions. *Queueing Systems. Theory and Applications*, **10**, 5–87.

3. Abate, J. and Whitt, W. (1992). Numerical inversion of probability generating functions. *Operations Research Letters*, **12**, 245–251.

4. Abate, J. and Whitt, W. (1992). Solving probability transform functional equations for numerical inversion. *Operations Research Letters*, **12**, 275–281.

5. Abate, J. and Whitt, W. (1995). Numerical inversion of Laplace transforms of probability distributions. *INFORMS Journal on Computing*, **7**, 36–43.

6. Abate, J. and Whitt, W. (1998). Calculating transient characteristics of the Erlang loss model by numerical transform inversion. *Communications in Statistics. Stochastic Models*, **14**, 663–680.

7. Abate, J. and Whitt, W. (1999). Computing Laplace transforms for numerical inversion via continued fractions. *INFORMS Journal on Computing*, **11**, 394–405.

8. Abate, J. and Whitt, W. (1999). Infinite-series representations of Laplace transforms of probability density functions for numerical inversion. *Journal of the Operations Research Society of Japan*, **42**, 268–285.

9. Abate, J., Whitt, W., and Hill, M. (1995). Numerical inversion of Laplace transforms of probability distributions. *ORSA Journal on Computing*, **7**, 36–43.

10. Abdel-Hameed, M. (1995). Inspection, maintenance and replacement models. *Computers & Operations Research and their Application to Problems of World Concern*, **22**, 435–441.

11. Abramson, M. (2006). NOMADm software. URL `http://www.afit.edu/en/enc/Faculty/MAbramson/nomadm.html`.

12. Abramson, M., Audet, C., and Dennis, J., Jr. (2006). Nonlinear programming by mesh adaptive direct searches. *SIAG/Optimization Views-and-News*, **17**, 1–17.

13. Abramson, M. A. (2002). *Pattern Search Filter Algorithms for Mixed Variable General Constrained Optimization Problems.* Ph.D. Thesis, Rice University.

14. Abramson, M. A. (2005). Second-order behavior of pattern search. *SIAM Journal on Optimization*, **16**, 515–530.

15. Ahmed, N. U. and Schenk, K. F. (1978). Optimal availability of maintainable systems. *IEEE Transactions on Reliability*, **27**, 41–45.

16. Anderson, E., Bai, Z., Bischof, C., Blackford, S., Demmel, J., Dongarra, J., Croz, J. D., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. (1999). *LAPACK User's Guide.* SIAM, Philadelphia.

17. Arioli, M., Codenotti, B., and Fassino, C. (1996). The Padé method for computing the matrix exponential. *Linear Algebra and its Applications*, **240**, 111–130.

18. Ascher, H. (1968). Evaluation of repairable system reliability using the 'bad-as-old' concept. *IEEE Transactions on Reliability*, **17**, 103–110.

19. Ascher, H. and Feingold, H. (1984). *Repairable Systems Reliability.* Marcel Dekker, New York.

20. Audet, C. (1998). Convergence results for pattern search algorithms are tight. Technical Report TR98-24, Department of Computational and Applied Mathematics, Rice University.

21. Audet, C. and Dennis, J. (2000). Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization*, **11**, 573–594.

22. Audet, C. and Dennis, J. E. (2003). Analysis of generalized pattern searches. *SIAM Journal of Optimization*, **13**, 889–903.

23. Audet, C. and Dennis, Jr., J. E. (2004). Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, **To appear**.

24. Audet, C. and Dennis, Jr., J. E. (2004). A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, **14**, 980–1010.

25. Audet, C. and J.E. Dennis, J. (2004). Mesh adaptive direct search algorithms for constrained optimization. Technical Report TR04-02, Rice Unversity.

26. Aven, T. and Bergman, B. (1986). Optimal replacement times - a general set-up. *Journal of Applied Probability*, **23**, 432–442.

27. Baker, G. A. and Graves-Morris, P. (1996). *Padé Approximants.* Cambridge University Press, New York.

28. Barlow, R. and Hunter, L. (1960). Optimum preventive maintenance with repair. *Operations Research*, **8**, 90–100.

29. Barlow, R. and Proschan, F. (1967). *Mathematical Theory of Reliability.* John Wiley & Sons, New York.

30. Barlow, R. and Proschan, F. (1981). *Statistical Theory of Reliability and Life Testing: Probability Models.* Holt, Rinehart, and Winston, New York.

31. Ben-Ari, Y. and Gal, S. (1986). Optimal replacement policy for multicomponent systems: an application to a dairy herd. *European Journal of Operations Research*, **23**, 213–221.

32. Bertsekas, D. P. (1999). *Nonlinear Programming.* Athena Scientific, Belmont, MA.

33. Biswas, A., Sarkar, J., and Sarkar, S. (2003). Availability of a periodically inspected system, maintained under an imperfect-repair policy. *IEEE Transactions on Reliability*, **52**, 311–318.

34. Bobbio, A. and Trivedi, K. (1986). An aggregation technique for the transient analysis of stiff Markov chains. *IEEE Transactions on Computers*, **35**, 803–814.

35. Booker, A. J., Dennis, J. E., Jr., Frank, P. D., Serafini, D. B., Torczon, V., and Trosset, M. W. (1999). A rigorous framework for optimization of expensive function by surrogates. *Structural Optimization*, **17**, 1–13.

36. Brown, M. and Proschan, F. (1983). Imperfect repair. *Journal of Applied Probability*, **20**, 851–859.

37. Bruns, P. (2000). Optimal maintenance strategies for systems with partial repair options and without assuming bounded costs. Technical report, Faculty of Mathematical Sciences, University of Twente.

38. Burden, R. L. and Faires, J. D. (2000). *Numerical Analysis.* Brooks Cole, Boston.

39. Cassady, C., Maillart, L., Bowden, R., and Smith, B. (1998). Characterization of optimal age-replacement policies. In *Reliability and Maintainability Symposium.* Anaheim, CA, 170–175.

40. Cassady, C. R., Pohl, E. A., and Jin, S. (2004). Managing availability improvement efforts with importance measures and optimization. *IMA Journal of Management Mathematics*, **15**, 161–174.

41. Cha, J. H., Lee, S., and Mi, J. (2004). Comparison of steady system availability with imperfect repair. *Applied Stochastic Models in Business and Industry*, **20**, 27–36.

42. Chen, B. and Zadrozny, P. A. (2001). Analytic derivatives of the matrix exponential for estimation of linear continuous-time models. *Journal of Economic Dynamics and Control*, **25**, 1867–1879.

43. Cho, D. and Parlar, M. (1991). A survey of maintenance models for multi-unit systems. *European Journal of Operational Research*, **51**, 1–23.

44. Chung, C. and Flynn, J. (1995). A branch-and-bound algorithm for computing optimal replacement policies in k-out-of-n systems. *Operations Research*, **43**, 826–837.

45. Çinlar, E. (1977). *The Theory and Applications of Reliability*, chapter Shock and Wear Models and Markov Additive Processes. Academic Press, San Diego, New York, 193–214.

46. Clarke, F. H. (1983). *Optimization and Nonsmooth Analysis.* Wiley, New York. Reissued in 1990 by SIAM Publications, Philadelphia, as Vol. 5 in the series Classics in Applied Mathematics.

47. Coope, I. D. and Price, C. J. (2001). On the convergence of grid-based methods for unconstrained optimization. *SIAM Journal on Optimization*, **11**, 859–869.

48. Cox, S. M. (2004). *Hybrid Stochastic Models for Remaining Lifetime Prognosis.* Ph.D. Thesis, Air Force Institute of Technology.

49. Czajkiewicz, Z. J. (1985). Optimization of the maintenance process. *Simulation*, **44**, 137–141.

50. Davis, D. (1952). An analysis of some failure data. *Journal of American Statistics Association*, **47**, 113–150.

51. Dekker, R. (1996). Applications of maintenance optimization models: a review and analysis. *Reliability Engineering and System Safety*, **51**, 229–240.

52. Dubner, H. and Abate, J. (1968). Numerical inversion of Laplace transforms by relating them to the finite Fourier cosine transform. *Journal of the Association for Computing Machinery*, **15**, 115–123.

53. Ebrahimi, N. (1999). Stochastic properties of a cumulative damage threshold crossing model. *Journal of Applied Probability*, **36**, 720–732.

54. Esary, J. D., Marshall, A. W., and Proschan, F. (1973). Shock models and wear processes. *Annals of Probability*, **1**, 627–650.

55. Finkelstein, D. E. (2004). *Analytical Results for a Single-Unit System Subject to Markovian Wear and Shocks.* M.S. Thesis, Air Force Institute of Technology.

56. Flynn, J., Chung, C., and Chaing, D. (1988). Optimal replacement policies for a multicomponent reliability system. *Operations Research Letters*, **7**, 167–172.

57. Forsythe, G. E., Malcolm, M. A., and Moler, C. B. (1977). *Computer Methods for Mathematical Computations.* Prentice Hall, Englewood Cliffs, NJ.

58. Gagan L. Choudhury, David M. Lucantoni, and Ward Whitt (1994). Multidimensional transform inversion with applications to the transient M/G/1 queue. *The Annals of Applied Probability*, **4**, 719–740.

59. Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations.* Johns Hopkins University Press, Baltimore.

60. Hein, J. and Bull, M. (2003). Capability computing: Achieving scalability on over 1000 processors. Technical report, The University of Edinburgh. Edinburgh, Scotland, UK.

61. Higham, N. J. (2005). The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, **26**, 1179–1193.

62. Igaki, N., Sumita, U., and Kowada, M. (1995). Analysis of Markov renewal shock models. *Journal of Applied Probability*, **32**, 821–831.

63. Ioffe, A. (1981). Nonsmooth analysis: Differential calculus of nondifferentiable functions. *Transactions of the American Mathematical Society*, **255**, 1–55.

64. Ioffe, A. (1994). Nonsmoothness and nonconvexity in calculus of variations and optimal control. In *Proceedings of the 33rd Conference on Decision and Control.* Lake Buena Vista, FL, 3986–3991.

65. Jorgenson, D. W., McCall, J., and Radner, R. (1966). Optimal maintenance of stochastically failing equipment. Technical report, RAND Corporation.

66. Kharoufeh, J. (2003). Explicit results for wear processes in a Markovian environment. *Operations Research Letters*, **31**, 237–244.

67. Kharoufeh, J., Finkelstein, D., and Mixon, D. (2006). Availability of inspected systems subject to Markovian wear and shocks. Working paper, Department of Operational Sciences, Air Force Institute of Technology.

68. Kiessler, P. C., Klutke, G.-A., and Yang, Y. (2002). Availability of periodically inspected systems subject to Markovian degradation. *Journal of Applied Probability*, **39**, 700–711.

69. Kijima, M. and Nakagawa, T. (1992). Replacement policies of a shock model with imperfect preventive maintenance. *European Journal of Operational Research*, **57**, 100–110.

70. Klutke and Yang (2002). The availability of inspected systems subject to shocks and graceful degradation. *IEEE Transactions on Reliability*, **51**, 371–374.

71. Klutke, G., Wortman, M., and Ayhan, H. (1996). The availability of inspected systems subject to random deterioration. *Probability in the Engineering and Informational Sciences*, **10**, 109–118.

72. Kulkarni, V. G. (1996). *Modeling and Analysis of Stochastic Systems.* Chapman & Hall, London.

73. Kumaran, J., Mitchell, K., and van de Liefvoort, A. (2005). Approximating matrix-exponential distributions by global randomization. *Stochastic Models*, **21**, 1–25.

74. Lagarias, J. C., Reeds, J. A., Wright, M. H., and Wright, P. E. (1998). Convergence properties of the Nelder-Mead Simplex method in low dimensions. *SIAM Journal on Optimization*, **9**, 112–147.

75. Lai, C. D. and Xie, M. (2004). Stochastic aging and dependence for reliability. Technical report, Massey University: Institute of Information Sciences and Technology.

76. Leemis, L. (1995). *Reliability: Probabilistic Models and Statistical Methods.* Prentice Hall, Englewood Cliffs, NJ.

77. Levitin, A. V. (2002). *Introduction to the Design and Analysis of Algorithms.* Addison Wesley, Boston.

78. Lewis, R. M. and Torczon, V. (1999). Pattern search algorithms for bound constrained minimization. *SIAM Journal on Optimization*, **9**, 1082–1099.

79. Lewis, R. M. and Torczon, V. (2000). Pattern search methods for linearly constrained minimization. *SIAM Journal on Optimization*, **10**, 917–941.

80. Lewis, R. M. and Torczon, V. (2002). A globally convergent augmented Lagrangian pattern search algorithm for optimization with general constraints and simple bounds. *SIAM Journal on Optimization*, **12**, 1075–1089.

81. Liou, M. (1966). A novel method of evaluating transient response. In *Proceedings of the IEEE*, 1. Lexington, MA, 20–23.

82. Lotka, A. J. (1939). A contribution to the theory of self-renewing aggregates with special reference to industrial replacement. *Annals of Mathematical Statistics*, **10**, 1–25.

83. Luss, H. (1976). Maintenance policies when deterioration can be observed by inspection. *Operations Research*, **24**, 359–366.

84. Mallor, F. and Santos, J. (2003). Classification of shock models in system reliability. *Monografías del Semin. Matem. García de Galdeano*, **27**, 405–412.

85. Marie, R., Reibman, A., and Trivedi, K. (1987). Transient solutions of acyclic Markov chains. *Performance Evaluation*, **7**, 175–194.

86. Marsaglia, G. and Zaman, A. (1991). A new class of random number generators. *The Annals of Applied Probability*, **1**, 462–480.

87. McCall, J. (1965). Maintenance policies for stochastically failing equipment: A survey. *Management Science*, **11**, 493–524.

88. McKinnon, K. I. M. (1999). Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, **9**, 148–158.

89. Meyer, C. D. (2001). *Matrix Analysis and Applied Linear Algebra.* Society for Industrial & Applied Mathematics, Philadelphia.

90. Michalewicz, Z. and Fogel, D. B. (2004). *How to Solve It: Modern Heuristics.* Springer, Berlin.

91. Mitra, D., Romeo, F., and Sangiovanni-Vincentelli, A. (1986). Convergence and finite-time behavior of simulated annealing. *Advances in Applied Probability*, **18**, 747–771.

92. Moler, C. and Van Loan, C. F. (2003). Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, **45**, 3–49.

93. Murdock, W. (1995). *Component Availability for an Age Replacement Preventive Maintenance Policy.* Ph.D. Thesis, Department of Industrial and Systems Engineering, Virginia Polytechnic Institute and State University.

94. Nachlas, J. A. (1989). Availability distribution based preventive-maintenance strategies. *International Journal of Modeling and Simulation*, **9**, 49–52.

95. Nachlas, J. A. and Cassady, C. R. (1999). Preventive maintenance study: A key component in engineering education to enhance industrial productivity and competitiveness. *European Journal of Engineering Education*, **24**, 299–309.

96. Nakagawa, T. (1979). Replacement problem of a parallel system in random environment. *Journal of Applied Probability*, **16**, 203–205.

97. Nakagawa, T. and Osaki, S. (1974). Optimum replacement policies with delay. *Journal of Applied Probability*, **11**, 102–110.

98. Nelder, J. A. and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, **7**, 308–313.

99. Neuts, M. F. (1995). *Algorithmic Probability: A Collection of Problems.* Chapman & Hall, London.

100. Newby, M. (1998). Analysing crack growth. *Journal of Aerospace Engineering*, **212**, 157–166.

101. Olsson, D. M. and Nelson, L. S. (1975). The Nelder-Mead Simplex procedure for function minimization. *Technometrics*, **17**, 45–51.

102. Osaki, S. and Nakagawa, T. (1976). Bibliography for reliability and availability of stochastic systems. *IEEE Transactions on Reliability*, **25**, 284–287.

103. Pieraccini, S. (2002). Hybrid Newton-type method for a class of semismooth equations. *Journal of Optimization Theory and Applications*, **112**, 381–402.

104. Pierskalla, W. and Voelker, J. (1976). A survey of maintenance models: The control and surveillance of deteriorating systems. *Naval Research Logistics Quarterly*, **23**, 353–388.

105. Press, W., Flannery, B., Teukolsky, S., and Vetterling, W. (1988). *Numerical Recipes in C.* Cambridge University Press, New York.

106. Qi, L. and Sun, J. (1993). A nonsmooth version of Newton's method. *Math. Program.*, **58**, 353–367.

107. Råde, J. (1976). Reliability in a random environment. *Journal of Applied Probability*, **13**, 407–410.

108. Rangan, A. and Sarada, G. (1992). Some results on the life distribution properties of systems subject to shocks and general repair. *Microelectronics Reliability*, **33**, 1–6.

109. Reineke, D., Murdock, W., Pohl, E., and Rehmert, I. (1999). Improving availability and cost performance for complex systems with preventive maintenance. In *Proceedings of the Annual Reliability and Maintainability Symposium.* Washington, DC, 383–388.

110. Ross, S. M. (1981). Generalized Poisson shock models. *Annals of Probability*, **9**, 896–898.

111. Ross, S. M. (2002). *Introduction to Probability Models.* Academic Press, New York.

112. Salamanca, H. and Quiroz, L. (2005). A simple method of estimating the maintenance cost of airframes. *Aircraft Engineering and Aerospace Technology*, **77**, 148–151.

113. Sandve, K. and Aven, T. (1999). Cost optimal replacement of monotone, repairable systems. *European Journal of Operational Research*, **116**, 235–248.

114. Shaked, M. and Zhu, H. (1992). Some results on block replacement policies and renewal theory. *Journal of Applied Probability*, **29**, 932–946.

115. Shanthikumar, J. G. and Sumita, U. (1983). General shock models associated with correlated renewal sequences. *Journal of Applied Probability*, **20**, 600–614.

116. Shelef, R. (1987). *New Numerical Quadrature Formulas for Laplace Transform Inversion by Bromwichs Integral.* M.S. Thesis, Israel Institute of Technology.

117. Sherif, Y. and Smith, M. (1981). Optimal maintenance models for systems subject to failure - A review. *Naval Research Logistics Quarterly*, **28**, 47–74.

118. Shue, S. H. (1993). A generalized model for determining optimal number of minimal repairs before replacement. *European Journal of Operational Research*, **69**, 38–49.

119. Singpurwalla, N. (1995). Survival in dynamic environments. *Statistical Science*, **10**, 86–103.

120. Sipe, J. A. (2003). *Transient Analysis and Applications of Markov Reward Processes*. M.S. Thesis, Air Force Institute of Technology.

121. Skiena, S. S. (1997). *The Algorithm Design Manual*. Springer, Berlin.

122. Smith, W. (1958). Renewal theory and its ramification. *Journal of the Royal Statistics Society*, **20**, 243–302.

123. Sobczyk, K. (1986). Modelling of random fatigue crack growth. *Engineering Fracture Mechanics*, **24**, 609–623.

124. Spears, W. M. (1998). A compression algorithm for probability transition matrices. *SIAM Journal on Matrix Analysis and Applications*, **20**, 60–77.

125. Speijker, L., van Noortwijk, J., Kok, M., and Cooke, R. (2000). Optimal maintenance decisions for dikes. *Probability in the Engineering and Informational Sciences*, **14**, 101–121.

126. Srinivasan, V. S. and Ramachandra, K. V. (1978). An application of Kalman techniques to estimating availability. *IEEE Transactions on Reliability*, **27**, 46–48.

127. Stephens, C. P. and Baritompa, W. (1998). Global optimization requires global information. *Journal of Optimization Theory and Applications*, **96**, 575–588.

128. Stewart, W. J. (1995). *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, NJ.

129. Sumita, U. and Shanthikumar, J. (1985). A class of correlated cumulative shock models. *Advances in Applied Probability*, **17**, 347–366.

130. Torczon, V. (1997). On the convergence of pattern search algorithms. *Optimization for the Society for Industrial and Applied Mathematics*, **7**, 1–25.

131. Valdez-Flores, C. and Feldman, R. (1989). A survey of preventive maintenance models for stochastically deteriorating single-unit systems. *Naval Research Logistics*, **36**, 419–446.

132. Valkó, P. P. and Abate, J. (2005). Numerical inversion of 2-D Laplace transforms applied to fractional diffusion equations. *Applied Numerical Mathematics*, **53**, 73–88.

133. van den Bosch, P., Dietz, D., and Pohl, E. (1999). Choosing the best approach to matrix exponentiation. *Computers and Operations Research*, **26**, 871–882.

134. Van Loan, C. F. (1975). A general matrix eigenvalue algorithm. *SIAM Journal on Numerical Analysis*, **12**, 819–834.

135. Van Loan, C. F. (1977). The sensitivity of the matrix exponential. *SIAM Journal on Numerical Analysis*, **14**, 971–981.

136. Vaurio, J. K. (1999). Availability and cost functions for periodically inspected preventively maintained units. *Reliability Engineering and Systems Safety*, **63**, 133–140.

137. Vinoy, K. and Jha, R. (1996). *Radar Absorbing Materials : From Theory to Design and Characterization*. Springer, Berlin.

138. Wang, H. (2002). A survey of maintenance policies of deteriorating systems. *European Journal of Operations Research*, **139**, 469–489.

139. Wang, H. and Pham, H. (1996). Optimal maintenance policies for several imperfect repair models. *International Journal of Systems Science*, **27**, 543–549.

140. Ward, R. C. (1977). Numerical computation of the matrix exponential with accuracy estimate. *SIAM Journal on Numerical Analysis*, **14**, 600–610.

141. Weeks, W. T. (1966). Numerical inversion of Laplace transforms using Laguerre functions. *Journal of the Association for Computational Machinery*, **13**, 419–429.

142. Weibull, W. (1939). A statistical theory of the strength of materials. *Ing Vetenskaps Akad Handl*, **151**, 1–45.

143. Weiss, M. A. (1998). *Data Structures and Algorithm Analysis in C++*. Addison Wesley, Boston.

144. Wortman, M., Klutke, G.-A., and Ayhan, H. (1994). A maintenance strategy for systems subjected to deterioration governed by random shocks. *IEEE Transactions on Reliability*, **43**, 439–445.

145. Wortman, M. A. and Klutke, G.-A. (1994). On maintained systems operating in a random environment. *Journal of Applied Probability*, **31**, 589–594.

146. Zuckerman, D. (1980). Inspection and replacement policies. *Journal of Applied Probability*, **17**, 168–177.

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 074-0188*

1. REPORT DATE *(DD-MM-YYYY)* 07-03-2006	**2. REPORT TYPE** **Master's Thesis**	**3. DATES COVERED** *(From – To)* Mar 2005 – Mar 2006

4. TITLE AND SUBTITLE OPTIMAL PERIODIC INSPECTION OF A STOCHASTICALLY DEGRADING SYSTEM	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) Booher, Timothy, B., Captain, USAF	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Street, Building 642 WPAFB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GOR/ENS/06-04
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSOR/MONITOR'S ACRONYM(S)**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This thesis develops and analyzes a procedure to determine the optimal inspection interval that maximizes the limiting average availability of a stochastically degrading component operating in a randomly evolving environment. The component is inspected periodically, and if the total observed cumulative degradation exceeds a fixed threshold value, the component is instantly replaced with a new, statistically identical component. Degradation is due to a combination of continuous wear caused by the component's random operating environment, as well as damage due to randomly occurring shocks of random magnitude. In order to compute an optimal inspection interval and corresponding limiting average availability, a nonlinear program is formulated and solved using a direct search algorithm in conjunction with numerical Laplace transform inversion. Techniques are developed to significantly decrease the time required to compute the approximate optimal solutions. The mathematical programming formulation and solution techniques are illustrated through a series of increasingly complex example problems.

15. SUBJECT TERMS

Markov process, wear process, shock process, reliability, availability, stochastic model, compound damage process, pattern search, availability optimization

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Jeffrey P. Kharoufeh, (ENS)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			**19b. TELEPHONE NUMBER** *(Include area code)*
U	U	U	UU	142	(937) 255-3636, ext 4603; e-mail: Jeffrey.Kharoufeh@afit.edu