THE UNIVERSITY OF CALGARY

Optimized Inspection Times of Systems

in

Stochastic Decay

by

Thomas Boyes Morrison

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MATHEMATICS AND STATISTICS

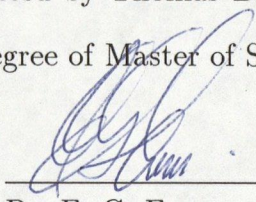
CALGARY, ALBERTA

April, 1993

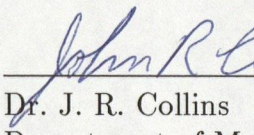
THE UNIVERSITY OF CALGARY

FACULTY OF GRADUATE STUDIES

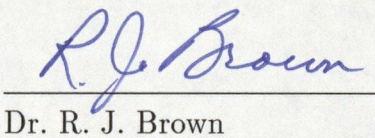
The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled, "Optimized Inspection Times of Systems in Stochastic Decay," submitted by Thomas Boyes Morrison in partial fulfillment of the requirements for the degree of Master of Science.



Dr. E. G. Enns
Department of Mathematics and Statistics



Dr. J. R. Collins
Department of Mathematics and Statistics



Dr. R. J. Brown
Department of Geology and Geophysics

Date April 22, 1993

Abstract

Dynamic programming techniques are used to examine the problem of determining inspection times of a system that can be modelled as a Markov process. The computer algebra system MACSYMA is used as the programming language. Recommendations for future work on determining optimized inspection times for more general systems than were examined in this thesis are documented.

Acknowledgements

A lot of friends have contributed to my being able to write this thesis and successfully attend graduate school. Without the support of any one person of the following the project would have been unimaginably more difficult.

To Robert Worthingham of the Department of Design Engineering of Nova Corporation of Alberta I owe my greatest thanks upon the completion of this project. Bob has supported me financially through several projects dealing with exterior corrosion growth on high pressure line pipe, and is responsible for introducing me to the problem of determining optimized inspection times. It was Bob's idea that I try to return to school after many years in the work force. His moral support since we met nearly 20 years ago, in particular since 1989 when I started working on the series of projects, has directly led to my success in graduate school.

Dr. Ernest G. Enns has given me much moral support during the two or so years we have known each other during the course of this project. Dr. Enns also provided funds for a wonderful trip to the Statistical Society of Canada convention in Edmonton during May 1992 and unexpectedly and most graciously provided funds for travel to the International Symposium on Symbolic and Computer Algebraic Computation (ISSAC '92) at the University of California, Berkeley in July 1992, and the Statistical Society of Canada convention at Acadia University in June 1993. A large amount of the fun of this project and being a graduate student has been due to Dr. Enns. While Bob Worthingham is responsible for my beginning this project, the assistance and support of Dr. Enns is responsible for my completing this project and being proud of the work.

Halsey Boyd has taught me how to be a student again and has convinced me that panic is not a viable methodology for completing assignments. During the very difficult time when my code could not match the results published in a paper Halsey helped me go through my code line by line and helped convince me that what I had programmed was indeed what the authors had written. Few people are as fortunate as I was to have the help of a close friend also experiencing the "joys" of graduate school.

Mike Nemeth has kept a close and watchful eye on my sanity during the past 21 months, and has provided financial and sanity support in the form of making sure I was always prepared for the next day's work. His sense of humour during the project has been most valuable. Mike has lived this project with me on a nearly day-by-day basis and without his help the probability of success would have been much lower. While the thesis examines optimized inspection times of systems, his intervention provided optimized maintenance of me before I broke and had to be replaced.

Alan Trigg and the Facility Integrity Group of the Field Services Department of Nova Corporation of Alberta have contributed financial and moral support to the work herein by hiring Morrison Scientific to determine whether corrosion pits from in-line-inspections of their pipelines can be automatically correlated from one inspection to the next. Some of the preparatory work for this NOVA work is examined in Chapters 7 and 8 of this thesis.

Bob Marcellus of C.M.E.L. Enterprises Ltd. has given me support since 1981. Bob has told me he always believed I had the capability to get an advanced degree. Here is the proof. The many projects I have had a part in while at C.M.E.L. were a fine training ground for the type of work required for preparing a thesis.

Lloyd Rankin of Mount Royal College offered much moral support and enthusiasm for my returning to graduate school. Without his support it would have been much more difficult to make those important decisions two years ago.

Mike Paolucci helped me understand how to calculate the logarithm of a matrix. His phone call and letter explaining how to do it were most welcome and provided insight as to how to turn a discrete Markov chain into an equivalent continuous-time Markov chain. How to do this is virtually impossible to find in the literature.

Ralph Klassen provided me financial support for four years from 1988 to 1992 and directly contributed to my starting my own company in Calgary. The foundation upon which this thesis was built was constructed by Ralph.

Dave Edwards contributed to my well being with moral support throughout the duration of this project, and checking of code during the difficult period of this project. Indy Lagu provided moral support and advice as the project neared its end.

At the completion of this project my supervisor, Dr. E.G. Enns, and the other members of my committee, Dr. J.R. Collins and Dr. R.J. Brown, made many constructive comments and made this thesis a better paper.

Finally, I would like to thank several other friends who have quietly asked "how school was going" every single week and have lived the highs and lows of the project with me. After over 90 weeks I can tell Margaret Kiedynk, Mike Nemeth and Jonathan Levine and Craig Friedt and the other members of Tuesday Night Mental Health Committee, Rosemary Snead and my running friends David McInnis and Joyce Stach that the project is complete and successful.

Dedication

I began running two years ago. I entered graduate school less than two years ago. Throughout this period I have been motivated by my silent friends Mabel C. Bragg and Watty Piper who wrote and retold, respectively, their wonderful stories "The Pony Engine" and "The Little Engine That Could." The stories are now over 60 years old and perhaps still serve as the first literature a child can read and understand where the objective of the protagonist is to reach some impossible goal. Each time I have been faced with difficulties running or being in graduate school I have remembered the wonderful phrase "I think I can. I think I can."

I dedicate this thesis to the memories of Mabel C. Bragg and Watty Piper, and to my mother Pat, my father Tom and my sister Sheila who also live by this unspoken family motto.

Contents

List of Figures

Chapter 1

Introduction and Motivation

In this thesis two methods for determining the inspection times of a system operating under stochastic[1] deterioration are examined. The project was motivated by several studies the author has conducted for the Department of Design Engineering, and the Facility Integrity Group of the Field Services Department of NOVA Corporation of Alberta. The studies began with a short review, or translation from "mathematics" into "english", of a paper describing a mathematical model that was used to describe the reliability of a decaying system [35, 48, 49]. The project developed into the implementation of the model for the Department of Design Engineering so they could use the model as part of their cost estimating procedures. Specifically, the model was used to examine exterior corrosion growth on high pressure natural gas pipelines. The idea was that the growth of corrosion on the outside of the pipeline could be modelled as being due to the quality of coating, described by a one or two element vector input by the user. A high quality coating reduces the corrosion growth to very small levels while a lesser quality coating does not inhibit growth and the reliability of the pipeline as a function of time is much less [36, 37].

The model prepared for the Department of Design Engineering was a brute-force model where a large system of differential equations was solved using Runge-Kutta techniques. Among the several options the user could select were those to determine a single time to inspect the pipeline such that the probability of failure (or repair)

[1] satisfying the laws of probability

of the pipeline would be a minimum over a preselected lifetime. The user could also select to determine two "optimized" inspection times. It was during the course of this work that the question arose as to whether it was possible to determine the next time to inspect the pipeline if its present state were known, over a not predetermined lifetime (i.e., over an infinite horizon). It is the determination of inspection times over the infinite horizon, motivated by the NOVA projects, that formed the problem to be solved in this thesis.

Given the differential equation model it remained to determine approximate values for the transition parameters [5].

Further motivation for solving the problem in this thesis was generated by other work dealing with analyzing corrosion growth data and, based on the data, estimating the reliability of the pipeline as a function of time. In this work, performed for the Facility Integrity Group of the Field Services Department of NOVA [6], measurements from two inspections of a pipeline were analyzed with the goal being to determine the rate of corrosion growth on the pipeline. The work is referred to as the "automatic correlation" problem because the idea is to write software that, given two inspection data sets, will automatically generate a Markov chain transition matrix for the growth of the corrosion between the two inspections. As preparation for this work, some of the problems have been analyzed in this thesis.

1.1 Outline of this Thesis

The work conducted for this thesis is presented in seven parts beginning with the next chapter.

In Chapter 2 the results of a literature search on the optimized inspection timing problem are presented.

The actual calculations of optimized inspection times are presented in Chapters 3, 5 and 6 with Chapter 4 giving some necessary preparation.

In Chapter 3 the simplest model of a deteriorating system is examined. Say the system has $N + 1$ states and the system can be in any state from 0 to N and still operate. In this model the ith state can only decay into the $i + 1$st state. The ultimate state is the failed state. The objective is to determine, given that the system is observed to be in state i, the time of next inspection such that the probability of the system having failed (or gone into the failed state) is kept below some chosen level α. No cost considerations are included in this model. It is assumed that the decay rates are known by the user.

A review of the dynamic programming methodology is presented in Chapter 4. This serves as an introduction to the more complicated dynamic programming models presented in Chapters 5 and 6.

Three of the models created by Mine and his co-workers in Japan during the 1970's and 1980's were implemented in software. Results from two of the models and their output are described in Chapters 5 and 6. These models use dynamic programming methodologies and illustrate what can be done solving the optimized inspection time problem from a theoretical viewpoint in contrast to the practical viewpoint of the final chapters. The model in Chapter 5 uses policy iteration; the model in Chapter 6 uses successive approximations. In contrast to the models presented in Chapters 3 and 4, cost considerations are used to determine optimized inspection times in the models considered in Chapters 5 and 6.

In Chapter 7 the problem of determining the decay rates is examined in detail. Given the transition matrix created by analyzing the system at two different times, the deterioration regime of the system must be determined. This is not a trivial problem. The objective of the work conducted for this chapter is to begin with a data set and end with a calculation of the failure probability of the system as a function of time into the future. Among other difficulties presented to someone trying to conduct the analysis for this chapter is the changing of the data from the discrete time Markov chain data into a continuous-time Markov chain for use in solving a system of differential equations. The greatest effort is concerned with obtaining the logarithm of a matrix. Only probabilities enter the calculations in this chapter. No cost considerations are involved.

Several other aspects of the optimized inspection time problem that are important to real-world systems are reviewed in Chapter 8. As well, for completeness, and to show what problems still must be solved for real-world situations, future work and recommendations are also reviewed in this chapter.

Chapter 2

Literature Search: The State-of-the-Art

The objective of this work is to determine inspection times over an infinite horizon. The literature obtained during the course of the research conducted for this project can be divided into four groups, or:

1. The exact knowledge situation, as documented by Mine and his group of researchers from Japan.

2. The false positive/negative situation, as documented by Pliska and his group of researchers.

3. Individual contributions from several researchers, mostly in the form of only a single paper, in contrast to the several contributions from the two previous groups.

4. Other interesting contributions.

There is a great difference in the models depending upon whether the failure of the system is immediately obvious to the researcher. The Mine and Pliska groups use models where failure is obvious. Many of the other researchers use models where failure is only known by inspection.

2.1 The Work of the Mine Group

Mine and his co-workers have produced several papers examining the problem of determining optimized inspection times. In each of the papers they use the same model for the deterioration and failure of the system. The failure of the system is obvious to the researcher. The topics of the papers are:

1. Mine and Kawai, 1974 [30]: Description of the deterioration model.

2. Mine and Kawai, 1975 [31]: Determination of optimized inspection times using minimum average cost per unit time as the criterion. Operating and repair costs do not depend on the state.

3. Mine and Kawai, 1976 [32] : Determination of optimized inspection time using minimum average cost per unit time as the criterion when the inspection interval is governed by a probability function, e.g. exponential distribution.

4. Mine and Kawai, 1982 [33]: Determination of optimized inspection times using minimum total discounted time when the system is not operating as the criterion.

5. Kawai, 1983 [15, 16]: Determination of ordering and replacement times using minimum average cost per unit time as the criterion, when the system is observed continuously.

6. Kawai, 1984 [17]: Determination of optimized inspection times using minimum expected total discounted cost as the criterion.

7. Ohnishi, Kawai and Mine, 1984 [39]: determination of optimized inspection times using minimum average cost per unit time as the criterion. The information about the state of the system is not exact.

8. Ohnishi, Kawai and Mine, 1986 [40]: Determination of optimized inspection times using minimum average cost per unit time as the criterion. Operating and repair costs depend on the state.

In each of these papers either a *policy iteration* procedure or a *successive approximation* procedure is used to determine the inspection times.

An attempt has been made to develop software to duplicate the results for three of the papers, ([31, 17, 40]) in the above list. In Chapter 5 the model for Mine and Kawai [31] is presented. In Chapter 6 the model for Ohnishi, Kawai and Mine [40] is presented. The model for Kawai [17] is not presented in this thesis.

2.1.1 General Overview of the Mine Group Papers

In all of the papers the majority of the effort is spent determining a function the authors refer to as H or G. In general the function will be referred to as H. The function H depends on the criteria used to solve the problem.

Each paper rehashes a proof that the function H has at most one minimum, or a minimum at ∞.

The system is described as being in a state ranging between 0 and N. The failure state is $N+1$. For each state, except state N, there exist three alternatives: i) immediately replace the system; ii) inspect t years in the future or iii) inspect at ∞, i.e. let the system decay until failure. For state N the two alternatives are to

replace immediately or let the system decay. (This might not be correct for all of the papers, but the generality of three alternatives is correct.)

For each state the alternative giving the least cost is chosen. If H has a minimum, then the value of H at the minimum is compared with the replacement option. If H does not have a minimum, then the value of H at $t = \infty$ is compared with the replacement option. An inspection time is chosen if H has a minimum, and if the value of H at this minimum is less than the replacement option.

2.1.2 Mine and Kawai, 1975 [31]

This paper is examined in detail in Chapter 5.

This is the only paper of the group that uses policy iteration. A system of equations is solved repeatedly until the average cost per unit time is repeated in two successive steps.

The software gives the same basic form of results as documented in the paper; however, a table of results in the paper cannot be completely reproduced.

2.1.3 Ohnishi, Kawai and Mine, 1986 [40]

This paper is examined in detail in Chapter 6.

The method used is successive approximation. Each state has associated with it an operating cost per unit time. At each step the option minimizing the cost per unit time is solved successively for each state.

2.2 The Work of the Pliska Group

S. R. Pliska, his graduate students and co-workers approach the optimized inspection time problem in a very different manner than the Mine group. Again, the failure of the system is obvious to the researcher. The major difference between the two groups is that instead of the exact state of the system begin made known to an inspector after an inspection, combinations of false positives and false negatives are included in the model. Another theme in the papers is that only one corrective action can be made during the infinite time period. These models are far more appropriate to medically oriented problems than machine oriented problems. None of the Pliska group models were analyzed as part of this thesis. The topics of some of their papers are:

1. Milioni, A.Z., 1987 [27]: This work forms Milioni's Ph.D. thesis, supervised by Pliska. Two general methods are used to determine inspection times. The optimized inspection times are determined by minimizing the probability of failure. They can also be determined by minimizing the budget such that the probability of failure is bounded from above. The applications examine the medical screening situation where, for example, breast cancer or a post-operative infection is the failed state. In situations such as these the cost of failure cannot be measured with a monetary cost, so a catastrophic (infinite cost) procedure is also modelled.

2. Milioni, A.Z., and S.R. Pliska, 1988 [28, 29]: These two papers are reviews and extensions of Milioni's dissertation. In the first a binary (positive/negative) test with finite probability of a false positive is modelled. A single repair (or

corrective action) can occur during the life of the system (i.e. a single operation). They use dynamic programming to compute the minimum expected cost, and the cost is a function of the age of the system. In the second paper the catastrophic case is examined in detail.

3. Özekici, S., and T. Papazyan, 1988 [42]: In this paper the authors also examine the catastrophic failure case. Inspection models comparing number of inspections and probability of detection of positives are analyzed.

4. Özekici, S., and S.R. Pliska, 1991 [43]: The possibility of false positives and false negatives are modelled in this paper.

2.2.1 General Overview of the Pliska Group Papers

The flavour of the Mine papers is significantly different from the flavour of the Pliska papers. The major difference is that the former group is more concerned with engineering applications where a measurement system can yield an accurate assessment of the state of the system. In the case of a machine the system can be repaired an infinite number of times. In the latter set of papers medical testing applications are modelled. In these cases false results from tests are important and once the system has failed (or the patient has died or suffered horribly), the model terminates.

The Pliska types of analyses, incorporating both false test results and catastrophic failure into the models, yield much more complicated models to be solved than do the Mine papers.

2.3 Other Individual Contributions

Literature reviews of the maintenance problem are presented in [21, 38, 41, 45, 52, 60].

Several other researchers have contributed to the optimized inspection time problem, although it was not obvious in the literature search conducted for this project that they have published as much on the specific problems as have the Mine and Pliska groups.

Barlow and Proschan [3] (and other work referenced in their book) are usually acknowledged as one of the starting points for work on inspection times. In Chapter 4 of their book several maintenance policies are examined. The models reviewed by Barlow and Proschan are different from those above in that the system modelled can fail without the researcher so knowing. The objective is to minimize the time between failure of the system and detection of the failure by the researcher. A typical example of such a system is a ballistic missile. A system such as this has to be operable continuously; however, failure of a guidance component is not obvious to an observer.

A variant of the Barlow and Proschan work is presented by Beichelt [4]. Failure can only be detected after the next inspection. Minimax techniques are used.

Failure can only be detected by inspection in the model of Kaio and Osaki [12]. Checking time and imperfect inspections are analyzed in this paper.

Minimization of average cost per unit time is used by Kao [13] to determine optimized inspection times. Policy iteration is used. This paper is similar in tone to the Mine group work.

Kander [14] models a system where failure can only be determined by inspection.

The efficiency of modelling the system as having N levels compared to a two level system is examined. Various checking and time costs are used as the optimization criteria.

The number of checks per unit time is used by Keller [18] to model the optimized inspection time problem. The failure can only be determined by inspection. The calculus of variations is used to determine the optimum frequency of inspections. As with the Barlow and Proschan model, the duration of time between failure and detection is important. A more accurate result than this is given in a second paper by Keller [19]. A non-linear differential equation is solved for the optimum frequency of checking.

Luss' master's thesis work is reviewed in [25]. Several different models are solved by differentiation and by dynamic programming. The main emphasis is on including a non-zero inspection time.

Costs of occupancy are included in the optimized inspection time analysis in another paper by Luss [26]. The deterioration rate for each state is exponential (and equal). Inspections are at constant time intervals. The model assumes an infinite number of cycles between repairs.

Constant and increasing failure rates are used to model the system deterioration in a paper by Mokkapati and Venkata [34]. The maintenance plan described in this paper is applied to a coal mine power system where groups of equipment, all with different failure rates, are analyzed.

The minimization of the expected average cost per unit time is used as a criterion by Sengupta [50]. The information obtained during an inspection is not perfect. A delayed replacement is possible when the salvage value of a system is taken into

account. State occupancy costs are accounted for by revising the costs. The model then becomes a single-cycle problem.

A two-state system is modelled by Sernik and Marcus [51]. The machine produces material with or without inspection. The situation is modelled by the underlying probability distribution of the machine being in the good or bad state. Several examples are presented.

Two simplifying assumptions are incorporated into the work of Sherwin [53]. The first is that incorporating an equal conditional probability of failure between two inspections, given survival to the inspection, results in a near optimal schedule. The second is that the fraction of failures prevented by inspection is a function only of the interval-risk under failure-only maintenance. The analyses presented by Sherwin "tend to justify traditional schedules and urge better supervision of maintenance".

A gamma failure distribution is used to model deterioration in the work by Sim and Endrenyi [55]. The mean time to preventive maintenance under Poisson distributed failures is used as the optimization criterion.

Incomplete information is used by Sondik [56] to model the inspection problem. A detailed example to a two-state system is given in the paper.

Optimal reliability based inspection schedules are being implemented for offshore petroleum structures in the North Sea. In particular, crack length criteria and the ability to determine the presence of cracks in a structure are among the criteria used in minimizing the total expected cost of the structure over its lifetime. This work is reviewed by Sorensen et al. [57] and papers referenced therein.

2.4 Other Interesting Contributions

A Brownian motion developing in time with associated cost and inspection cost is analyzed in two large papers by Anderson and Friedman [1, 2].

Various applications to the medical screening situation are given in [20], [24], [46], [47] [54], and [61].

Kumar, Kapoor and Gupta [22] examine a series system using policy iteration.

Chapter 3

The Simplest Type of Inspection Problem

In this chapter some terminology will be defined and a simple model of a decaying system will be examined.

3.1 Introduction and Definitions: A Markov Chain

In all of the transition schemes reviewed here, in the simplest case, the probabilities that describe the state-to-state transitions do not change with time.

3.1.1 A Markov Process

The concept of a Markov process is very powerful. A sequence of states occupied by a system at times $t = n, t = n-1, \ldots, t = 0$ can be represented as $i, k, \ldots, m$ [11, page 3, Volume 1]. The researcher usually desires to know (or calculate) the probability that, after the next transition, the system will be in state j given that the system has visited states $i, k, \ldots, m$ in order, or

$$P\{s(n+1) = j | s(n) = i, s(n-1) = k, \ldots, s(0) = m\}.$$

Markov's assumption was that only the last state occupied by the system is necessary to calculate the future development, or

$$P\{s(n+1) = j | s(n) = i, s(n-1) = k, \ldots, s(0) = m\} = P\{s(n+1) = j | s(n) = i\}.$$

An english translation of Markov's 1907 paper is provided in Appendix B of Volume 1 of Howard [11].

In other words, a transition takes place at each time instant. The probabilities governing what state the process makes a transition to are given by the transition matrix. In order to calculate the future development of the process at any time, only the last state occupied by the process is relevant.

3.1.2 Discrete Semi-Markov Process

The system no longer makes a transition at every time instant. The duration the system is in any state is determined by an integer-valued random variable that depends on the state presently occupied and the state to which the transition will be made [11, page 577, Volume 2].

3.1.3 Continuous Semi-Markov Process

In the continuous semi-Markov process the transitions can occur at any time [11, page 687, Volume 2].

3.1.4 Continuous Markov Process

In the continuous-time Markov process only the state presently occupied by the system is relevant to its future development. The probability density function describing the duration the system is in a state does not depend on the state to which the system will go next. The length of time the system has been in a state is irrelevant both to estimating the state to which the system will go, and the remaining time the system will spend in the present state [11, page 769, Volume 2].

3.2 How Are the Values in the Continuous Transition Matrix Interpreted?

The discrete transition matrix is used to determine the probability $P_{ij}(0,t)$ the system will be in state j at time t given that it was in state i at time 0. As stated above, most often it is assumed that the values in the transition matrix do not depend on time.

The probability vectors and the general transition matrix are

$$\begin{pmatrix} p_1(t) \\ p_2(t) \\ \vdots \\ p_n(t) \end{pmatrix} = \begin{pmatrix} p_1(0) \\ p_2(0) \\ \vdots \\ p_n(0) \end{pmatrix}^T \begin{pmatrix} p_{11}(0,t) & p_{12}(0,t) & \dots & p_{1n}(0,t) \\ p_{21}(0,t) & p_{22}(0,t) & \dots & p_{2n}(0,t) \\ \vdots & \vdots & \ddots & \vdots \\ p_{m1}(0,t) & p_{m2}(0,t) & \dots & p_{mn}(0,t) \end{pmatrix}.$$

The vector on the left is the probability vector measured at time t. The vector on the right hand side is the probability distribution at the start of the simulation at time 0. The matrix on the right is the transition matrix.

Differential equations can used to solve the continuous problem. A typical example of continuous-time differential equations are

$$\begin{aligned} \frac{d}{dt} p_1(t) &= -\lambda_1 p_1(t) \\ \frac{d}{dt} p_k(t) &= -\lambda_k p_k(t) + \lambda_{k-1} p_{k-1}(t), \qquad k > 1. \end{aligned}$$

The differential equations are for a very simple system, namely that where the system can either stay in the same state (1 or k) or can move to the next larger state $(k+1)$. In the situation to be analyzed in Chapter 7 of this thesis the system can move from its present state to *any* higher state.

The coefficients of the differential equations can be represented in matrix fashion. The diagonal of the coefficients matrix has all negative values or zero's as elements. The negative values indicate the system is leaving that particular state. A zero value means the system remains in that state. All the off-diagonal elements are either zero or some positive number. For each row the sum of the off-diagonal elements equals the negative of the diagonal element.

In Chapter 7 the problem of estimating the elements of the transition matrix $p_{ij}(0,t)$ and the coefficients of the differential equations λ_k will be reviewed.

Knowledge of the decay regime is necessary before optimized inspection times can be determined. In the next section the simplest model of deterioration of a system will be examined.

3.3 Stochastic Deterioration of a Three-Level System

As an introduction to the inspection time problem, in this section the simplest possible inspection time problem will be examined.

The model examined here is a continuous-time Markov process.

In Figure 3.1 the transition diagram for a three (operating) state system is illustrated. The operating states are labeled 0, 1 and 2. The arrows between the states indicate that when the system is in state 0 the system can only decay to state 1, and when in state 1 the system can only decay to state 2, etc.

The rates of decay, assuming an exponential model, are given by

$$\lambda_0 = 1/200$$

$$\lambda_1 = 1/100$$

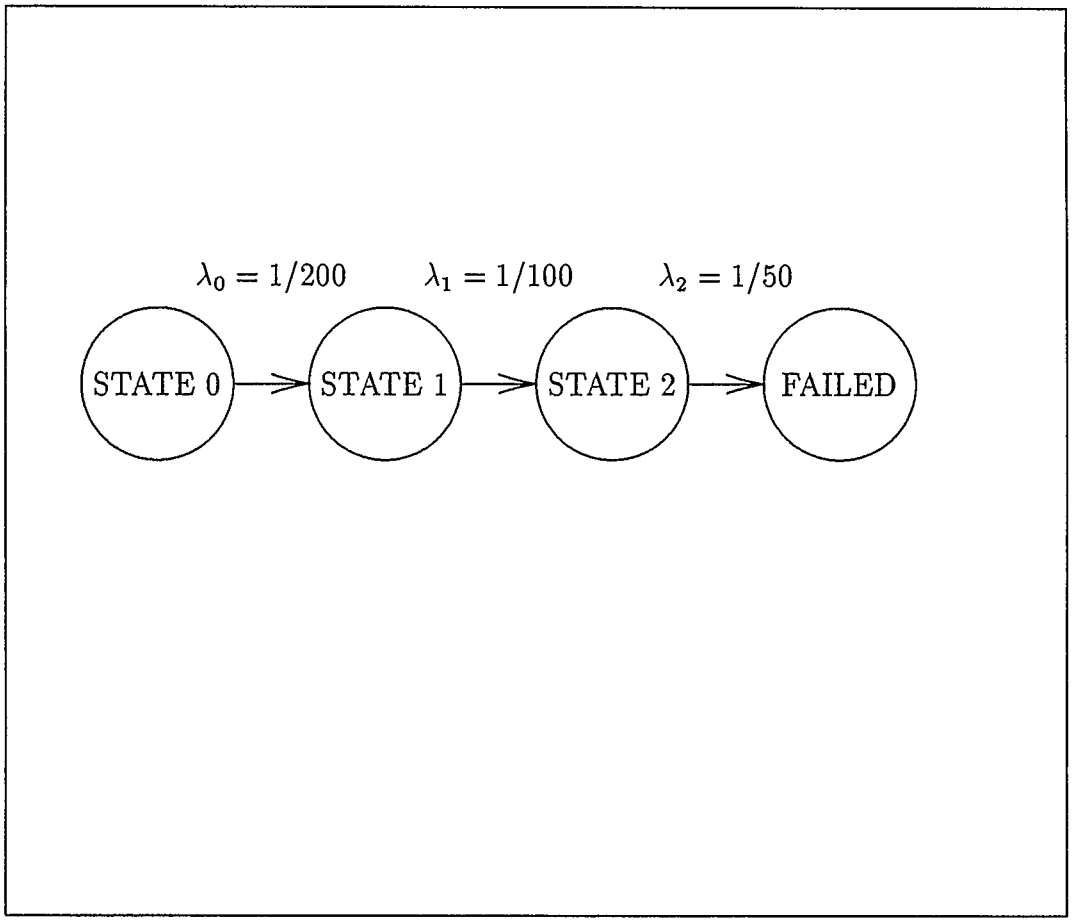


Figure 3.1: Transition Diagram for the Simplest Inspection Problem. The exponential transition parameters are given above the arrows.

$$\lambda_2 \;=\; 1/50$$

The notation above means that the system decays out of state 0 according to an exponential distribution with $\lambda_0 = 1/200$, or as $\frac{1}{200}e^{-t/200}$. The values are arbitrary; what is important for this simple model is that no two of the λ values are equal.

If the reader prefers, a different picture of the decay chain may make the introduction easier. The system shown in Figure 3.1 can model a radioactive decay chain if only those transitions shown are valid.

If knowledge of the system is obtained at some time, it is possible to calculate the probabilities of the system being in all states of greater decay at any time in the future. If none of the λ_i for each state are equal (as in this example), it is shown by Chiang ([7, page 215]) that the probability of the system being in State j, given that it is in State i at time zero is given by

$$\begin{aligned} P_{ij}(t) \;&=\; P\{\text{system in State } j \text{ at time } t | \text{system in State } i \text{ at time } 0\} \\ &=\; (-1)^{j-i}\,\lambda_i \ldots \lambda_{j-1} \left[\sum_{k=i}^{j} \frac{e^{-\lambda_k t}}{\prod\limits_{\substack{l=i \\ l \neq k}}^{j} (\lambda_k - \lambda_l)} \right]. \end{aligned}$$

This type of process is known as a *pure birth process*; each state has a different decay rate and the system can only decay to the next state. How to derive the above formula is shown by Chiang [7, page 215] and by Taylor and Karlin [59, pages 213–215].

In order to determine *when* the next inspection should occur the above formula

can be solved for any of the states 0, 1 or 2. These are cumulative probabilities.

Consider a situation where it is desired to maintain the probability of failure of the system below $\alpha = 0.1$.

If the system is inspected and is found to be in state 2, the researcher can determine that the cumulative probability of failure as a function of time is given by

$$\begin{aligned} P(\text{failure}) &= 1 - P_{22} \\ &= 1 - e^{-\lambda_2 t}. \end{aligned}$$

This function is illustrated in Figure 3.2.

Similarly, if the system is observed to be in state 1, the researcher can determine that the cumulative probability of failure as a function of time is given by

$$\begin{aligned} P(\text{failure}) &= 1 - P_{11} - P_{12} \\ &= 1 - e^{-\lambda_1 t} - (-1)^{2-1}\lambda_1 \left\{ \frac{e^{-\lambda_1 t}}{\lambda_1 - \lambda_2} + \frac{e^{-\lambda_2 t}}{\lambda_2 - \lambda_1} \right\}. \end{aligned}$$

This function is illustrated in Figure 3.3.

The cumulative probability of failure as a function of time if the system is observed to be in State 0 is given by

$$\begin{aligned} P(\text{failure}) &= 1 - P_{00} - P_{01} - P_{02} \\ &= 1 - e^{-\lambda_0 t} - (-1)^{1-0}\lambda_0 \left\{ \frac{e^{-\lambda_0 t}}{\lambda_0 - \lambda_1} + \frac{e^{-\lambda_1 t}}{\lambda_1 - \lambda_0} \right\} \\ &\quad -(-1)^{2-0}\lambda_0\lambda_1 \left\{ \frac{e^{-\lambda_0 t}}{(\lambda_0 - \lambda_1)(\lambda_0 - \lambda_2)} \right. \\ &\quad \left. + \frac{e^{-\lambda_1 t}}{(\lambda_1 - \lambda_0)(\lambda_1 - \lambda_2)} + \frac{e^{-\lambda_2 t}}{(\lambda_2 - \lambda_0)(\lambda_2 - \lambda_1)} \right\}. \end{aligned}$$

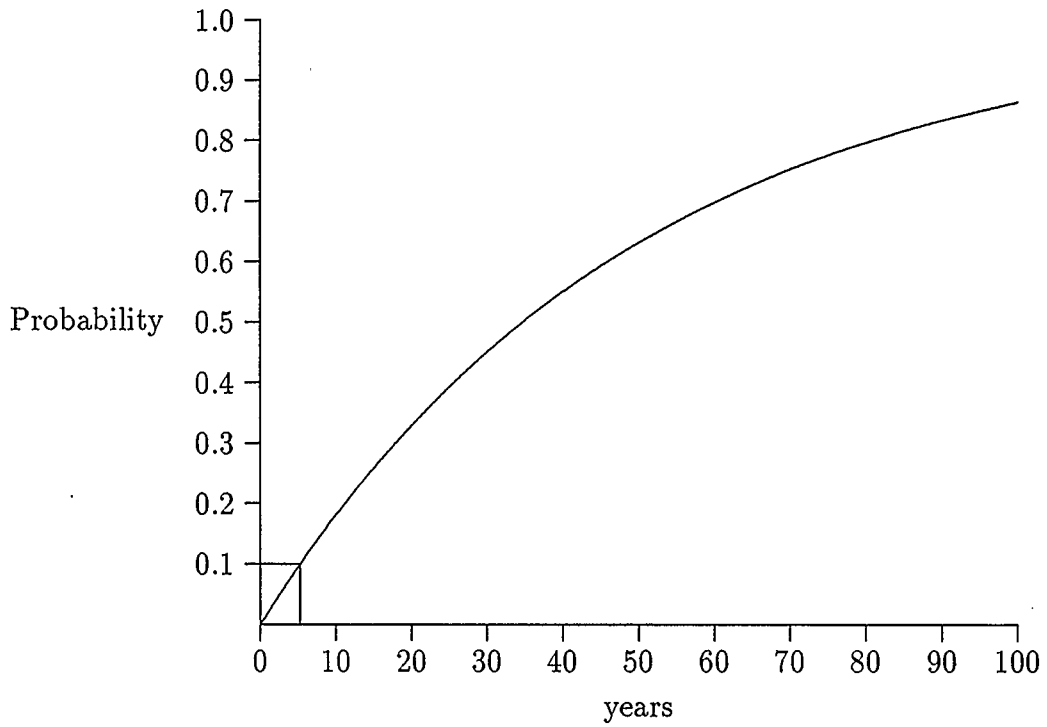


Figure 3.2: Inspection Time For State 2. If the system is Inspected to be in State 2, then the Cumulative Probability of Failure as a Function of Time is Given by the Curved Line. To Maintain a Cumulative Probability below 0.1 the System Must be Inspected Again before 5.26 Years.

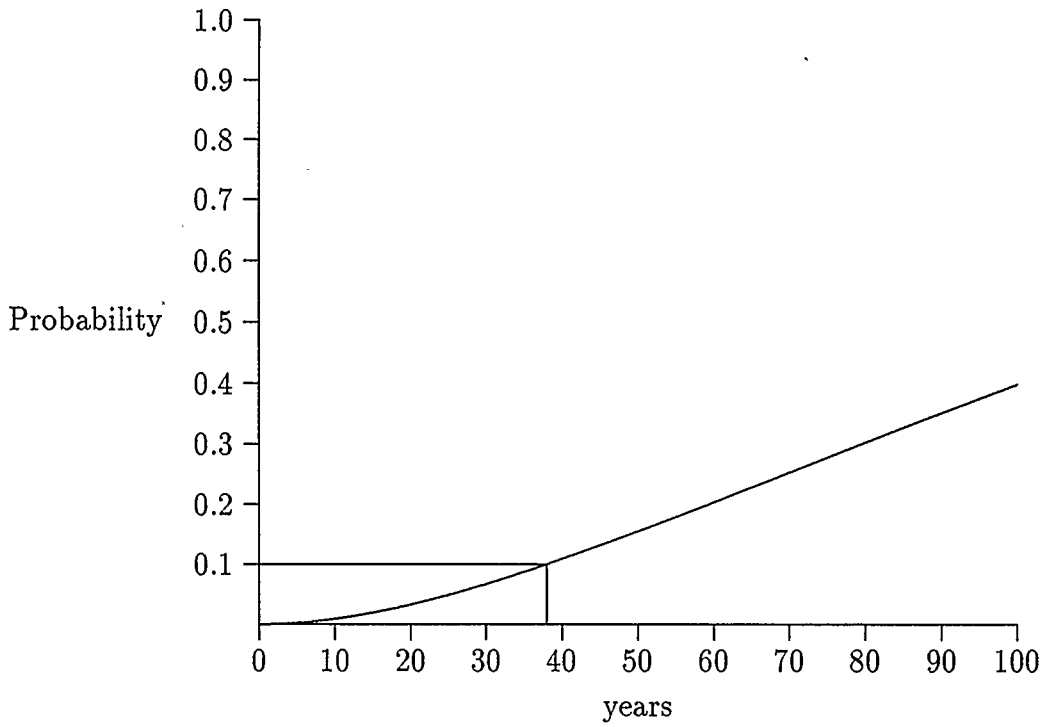


Figure 3.3: Inspection Time For State 1. If the system is Inspected to be in State 1, then the Cumulative Probability of Failure as a Function of Time is Given by the Curved Line. To Maintain a Cumulative Probability below 0.1 the System Must be Inspected Again before 38.01 Years.

This function is illustrated in Figure 3.4.

If observed in state 2 the next inspection should occur before or at 5.26 years. If observed in state 1 the next inspection should occur before or at 38.01 years, and if observed in state 0 the next inspection should occur before or at 76.06 years.

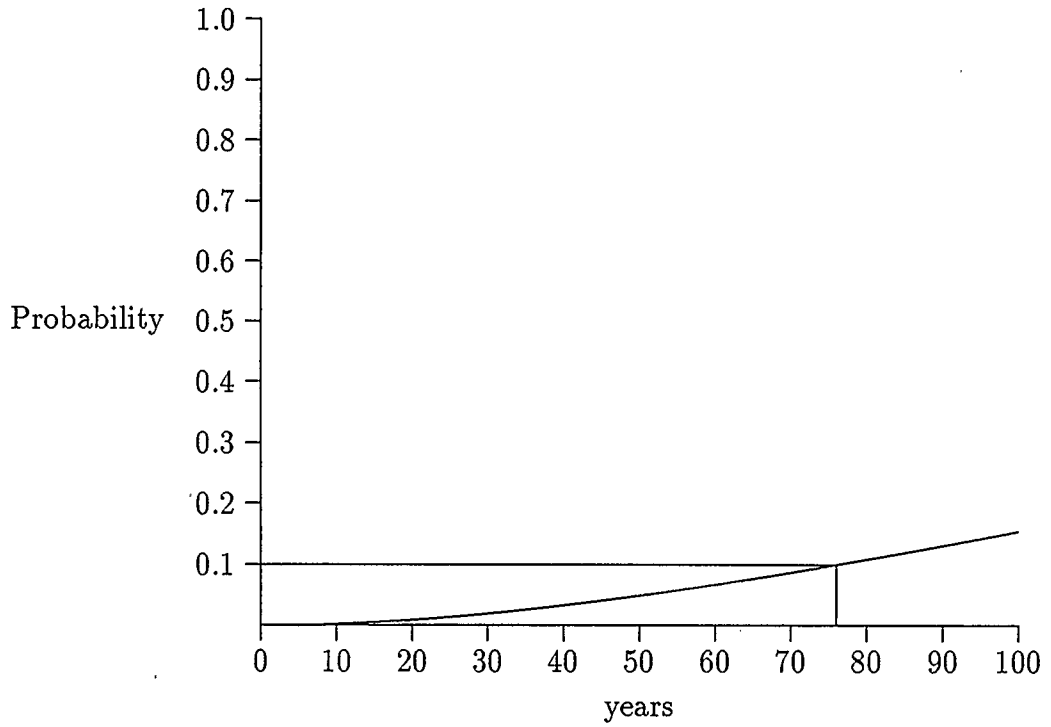


Figure 3.4: Inspection time For State 0. If the system is Inspected to be in State 0, then the Cumulative Probability of Failure as a Function of Time is Given by the Curved Line. To Maintain a Cumulative Probability below 0.1 the System Must be Inspected Again before 76.06 Years.

Chapter 4

Dynamic Programming

4.1 Introduction

Dynamic programming is a well established technique for solving decision problems when decisions can be made iteratively. The problem of examining a car rental agency will be reviewed in this chapter as an introduction to dynamic programming.

The dynamic programming technique consists of two steps.

To begin the problem an arbitrary set of decisions is created. At each state more than one alternative or decision can be made. The objective is to determine the optimal decision path for all states.

1. In the first step a single value for the entire set of decisions is determined. When a researcher is using the policy iteration technique the value is determined by solving a set of equations based on the decisions at each state for variables that depend upon the steps chosen. When a researcher is using the successive approximation technique a value is determined by solving a single equation for each state.

2. The effort for the second step has its beginning at the penultimate state. Based on the values determined in the above step, the decision yielding the maximum reward (or minimum cost) is determined. Then, using the chosen decision for the penultimate state, the decision yielding the maximum reward (or minimum

cost) for the second but penultimate state is determined. As an introduction to this and the next two chapters: In the problem examined in this chapter the objective is to determine the decisions such that the gain or reward per unit time on the rented cars is a maximum; in Chapters 5 and 6 the objective is to determine the decisions such that the costs between the present time and the next installation of a new system are minimized.

The two procedures are repeated until the decisions obtained at each state do not change from one iteration to the next.

The dynamic programming formulation is completely general. In order to introduce the reader to the process, and the method proposed for solving problems of this type, a simple problem from Howard [11, pages 983-993] will be examined.

4.2 The Car Rental Agency

Consider a car rental agency interested in maximizing the average reward per unit time over an infinite time frame. The objective is to determine how the car rental agency should operate in order to receive a maximum amount of money (gain or reward) per unit time.

The computer algebra system MACSYMA [58] will be used as the programming language here and elsewhere in this thesis. Variables and parameters in the program are denoted by italic type. MACSYMA functions and control structures (such as "for" loops) are denoted by typewriter type.

The parameters used in the problem DYNAMIC1 are:

- i is the state,

- k is an alternative, and where used below indicates that the parameter depends upon the alternative chosen,
- g is the gain or reward per unit time, to be maximized,
- ν_i the relative value of state i. The value of ν_2 is set to zero. The relative values are measured from a reference point of state 2, otherwise the system of equations would not equal the number of unknowns. The state with value set to zero is arbitrary.
- r_i^k is the expected reward per occupancy. The reward depends on the state and the alternative. The reward is known as r or *reward* in the MACSYMA program.
- $\bar{\tau}_i^k$ is the mean waiting time in each state. The waiting time is known as w or *waiting* in the MACSYMA program.
- q_i^k is the earning rate for each state, and is equal to $r_i^k/\bar{\tau}_i^k$,
- p_{ij}^k are the transition probabilities between states. The probability of transition from state i to state j depends upon alternative k. In the program the transition probabilities are called *transition_matrix*.
- Γ_i^k is the test quantity, used to determine the alternative for each state that maximizes the return.

$$\Gamma_i^k = q_i^k + \frac{1}{\bar{\tau}_i^k}\left[\sum_{j=i}^{N} p_{ij}^k \nu_j - \nu_i\right].$$

In the program the test quantity is known as *test_quan*.

A flow chart of the procedure is illustrated in Figure 4.1.

The MACSYMA program is given in Figure 4.2. The value determination routine is given in Figure 4.3 and the policy iteration routine is given in Figure 4.4.

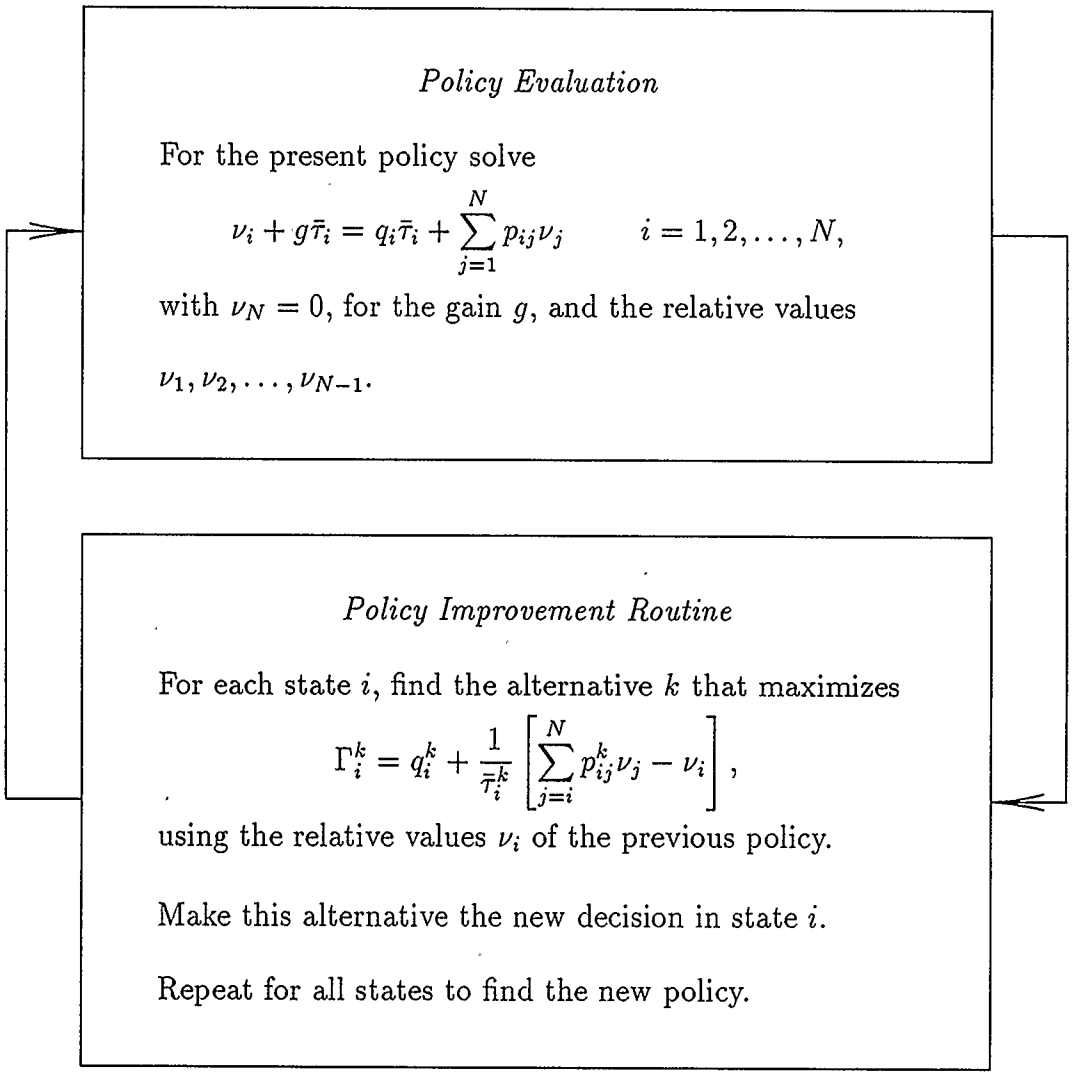


Figure 4.1: The Dynamic Programming Algorithm for the Car Rental Agency

```
array(r,2,2);
r[1,1]: 45 , r[1,2]: 90 $
r[2,1]: 60 , r[2,2]: 20 $

w  : matrix( [ 3.6, 6 ], [ 9.6, 4 ] )$

array(transition, 2,2,2);
transition[1,1,1] : 0.8 , transition[1,2,1] : 0.2 $
transition[1,1,2] : 0.0 , transition[1,2,2] : 1.0 $
transition[2,1,1] : 0.3 , transition[2,2,1] : 0.7 $
transition[2,1,2] : 1.0 , transition[2,2,2] : 0.0 $

states            : 2 $

alternative_max   : 2 $
array(alternative_start, alternative_max);
alternative_start[1] : 2;
alternative_start[2] : 2;

test_1            :  alternative_start $
test_2            : -alternative_start $

for i : 1 thru 100 while test_1 # test_2 do (
   test_2 : test_1,
   answer : equation_solve(states, r, transition, w, test_1),
   test_1 : improvement(   states, r, transition, w, v,
                        alternative_max));

g : part(answer,1), numer;

print(" g = ", g);

for i : 1 thru states do (
   t[i] : v[i],numer,
   print( " state ", i, " v =    ", t[i],
          " new policy =    ", test_1[i] ) );
```

Figure 4.2: The MACSYMA Program DYNAMIC1

```
/* EQUATION_SOLVE */

equation_solve(num_states, reward, transition_matrix, waiting,
               alternative) :=
block([ g, i, j, k, v, g_stuff, eqn, list_of_ans],

remvalue( g ),
for i : 1 thru num_states do (
   remvalue( v[i] ), g_stuff[i] : 0 );

v[num_states] : 0,

for i:1 thru num_states do
   for j:1 thru num_states do (
      k : alternative[i],
      g_stuff[i] : g_stuff[i] + transition_matrix[i,j,k] * v[j] ),

for i:1 thru num_states do (
   k : alternative[i],
   eqn[i]: v[i] + g * waiting[i,k] - reward[i,k] - g_stuff[i] ),

for i: 1 thru num_states do
   display(eqn[i],g_stuff[i]),

globalsolve: true,
linsolve([ eqn[1], eqn[2] ],[ g, v[1] ]),

list_of_ans:[ g, v[1] ],
return[list_of_ans])$

/* END OF FUNCTION EQUATION_SOLVE */
```

Figure 4.3: Policy Evaluation or Equation Solving Routine for the Program DYNAMIC1

```
/* IMPROVEMENT */

improvement(num_states, reward, transition_matrix, waiting, v,
            alternative_max) :=
     block([max_test, p_stuff, test_quan, alternative ],

for i:1 thru num_states do (
   for k:1 thru num_states do
   p_stuff[i,k]: 0),

for i: 1 thru num_states do (
   for k: 1 thru alternative_max do (
      for j: 1 thru num_states do (
         p_stuff[i,k] : p_stuff[i,k] +
                        transition_matrix[i,j,k] * v[j] ),
       test_quan[i,k] : ( reward[i,k] + p_stuff[i,k] - v[i] ) /
                        waiting[i,k]) ),

for i: 1 thru num_states do (
   for k: 1 thru alternative_max do
      print(i, k, p_stuff[i,k], test_quan[i,k])),

for i: 1 thru num_states do (
   max_test: -1000,
   for k: 1 thru alternative_max do (
      if test_quan[i,k] > max_test then (
         max_test : test_quan[i,k],
         alternative[i] : k ))),

return(alternative))$

/* END OF FUNCTION IMPROVEMENT */
```

Figure 4.4: Policy Iteration Routine for the Program DYNAMIC1

4.2.1 Overview of the Program DYNAMIC1

In Figure 4.2 the driver for the program is illustrated. At the start of the program the values for the reward, waiting time and transition parameters are initialized. The two variables *alternative_1* and *alternative_2* are the beginning set of alternatives. The program begins by setting *alternative_2* as the chosen alternative for both states. The two variables *test_1* and *test_2* are the two sets of alternatives before and after the iteration. They are deliberatly set to different values at the beginning of the program.

The `for` loop is implemented until *test_1* is equal to *test_2* (i.e., the alternatives on two iterations of the loop remain the same). An arbitrary maximum of 100 times through the loop is set. *test_2* is updated to equal *test_1* at the beginning of the loop. *test_1* is changed in the loop.

In the function `equation_solve` the equations for the gain (g) and the relative value of state 1 (ν_1) are first determined and then solved (Figure 4.3). *g_stuff*[*i*] is the sum $\sum_{j=1}^{N} P_{ij}\nu_j$. The equation *eqn*[*i*] is written in the form where the equation is set to zero (MACSYMA assumes the equation is to be set to zero if the user does not explicitly set the equation using an equal sign). The MACSYMA function `linsolve([ eqn[1], eqn[2] ],[ g, v[1] ])` solves the two equations for g and v[1].

In the function `improvement` the alternative maximizing *test_quan* for each state is determined.

The final result (the value of g and the value of ν_1) is printed out at the end of the program. *test_1* and *test_2* are compared as the loop counter is increased by 1 in

the MACSYMA program. If they are not equal the counter is increased by 1 and the routines `equation_solve` and `improvement` are enacted again. If *test_1* and *test_2* are equal the program stops.

4.2.2 The Value Determination (`Equation_Solve`) Routine

The MACSYMA command `remvalue` removes the value bounded to g, the ν's for each state, and the sum of the products of $p_{ij}\nu_j$, (g_stuff).

The equations are built up and solved using `linsolve`. The parameter `globalsolve : true` bounds the values of g and ν_1 such that they can be used in other places in the program and are not unique to the function. The `return` statement passes g and ν_1 to the main program where they can be used in the policy iteration routine.

4.2.3 The Policy Iteration (`Improvement`) Routine

All parameters are reset to zero values.

In the first `for` loop the test quantity is determined. The second `for` loop displays the values of the test quantity as a function of alternative and the third `for` loop determines the alternative that yields a maximum for each state. The alternatives are used as the new values for *test_1*.

Upon returning to the main program, *test_1* and *test_2* are compared to determine if it is necessary to perform the `equation_solve` and `improvement` routines again.

4.2.4 The MACSYMA Output for the Program DYNAMIC1

The output from the program is illustrated in Figures 4.5, 4.6 and 4.7. It is a relatively straightforward matter for an interested reader to follow the output.

In the first figure the equations to be solved are created in the routine `equation_solve`. *g_stuff* is printed out for debugging purposes. Upon leaving `equation_solve` the values of g (11) and ν_1 (24) are returned to the main program to be used in `improvement`.

In the second figure the `improvement` routine is entered. the values printed out are, in order, the state i, the alternative k, the sum $\sum_{j=i}^{N} p_{ij}^k \nu_j$ and the test quantity Γ_i^k. Note that the values for g and ν_1 from the previous iteration are recognizable in this printout. For state 1 the maximum test quantity occurs for *alternative_1*. For state 2 the maximum test quantity occurs for *alternative_2*. Since *test_1* is not equal to *test_2* the second iteration of the loop is done. New equations are created and solved. Upon leaving `equation_solve` the values of g (53521150/4805981 = 11.136) and ν_1 (117964980/4805981 = 24.545) are returned to the main program to be used in `improvement`.

In the third figure the alternatives maximizing the gain are determined as being *alternative_1* for state 1 and *alternative_2* for state 2. *test_1* and *test_2* are equal, so the program stops (prints DONE) and the values of g, ν_1 and the alternatives are printed.

The problem of maximizing the gain has been solved in two iterations.

```
(D1)   \MACSYMA\THESIS\dynamic.out
(C2) BATCH("C:\\MACSYMA\\THESIS\\DYNAMIC1.MAC");

REITERATION OF THE ROUTINES OMITTED FROM THE OUTPUT

(C29) for i : 1 thru 100 while test_1 # test_2 do (
    test_2 : test_1,
    answer : equation_solve(states, r, transition, w, test_1),
    test_1 : improvement(   states, r, transition, w, v,
                            alternative_max));

                                          [ 3.6  6 ]
1 Enter EQUATION_SOLVE [2, R, TRANSITION, [       ],
                                          [ 9.6  4 ]
        ALTERNATIVE_START]

                          EQN  = 6 G + V  - 90.0
                             1          1

                            G_STUFF  = 0.0
                                   1

                         EQN  = 4 G - 1.0 V  - 20
                            2             1

                            G_STUFF  = 1.0 V
                                   2        1


1 Exit  EQUATION_SOLVE RETURN
                                    [11, 24]
```

Figure 4.5: First Part of Output from MACSYMA

```
                                          [ 3.6  6 ]
1 Enter IMPROVEMENT [2, R, TRANSITION, [       ], V, 2]
                                          [ 9.6  4 ]
1 1  19.2  11.16666
1 2  0.0   11.0
2 1  7.2    7.0
2 2  24.0  11.0
1 Exit  IMPROVEMENT ALTERNATIVE

                                            [ 3.6  6 ]
1 Enter EQUATION_SOLVE [2, R, TRANSITION, [       ], ALTERNATIVE]
                                            [ 9.6  4 ]

                          EQN  = 3.6 G + 0.2 V  - 45
                             1                1

                              G_STUFF  = 0.8 V
                                     1        1

                          EQN  = 4 G - 1.0 V  - 20
                             2              1

                              G_STUFF  = 1.0 V
                                     2        1


1 Exit  EQUATION_SOLVE RETURN
                        [53521150/4805981, 117964980/4805981]
```

Figure 4.6: Second Part of MACSYMA Output

```
                                          [ 3.6  6 ]
1 Enter IMPROVEMENT [2, R, TRANSITION, [       ], V, 2]
                                          [ 9.6  4 ]
1 1  19.63635  11.13636
1 2  0.0       10.90909
2 1  7.36363    7.01704
2 2  24.54545  11.13636

1 Exit  IMPROVEMENT ALTERNATIVE

(D29)       DONE

(C30) g : part(answer,1), numer;
(D30)                          [11.13636, 24.54545]

(C32) for i : 1 thru states do (
   t[i] : v[i],numer,
   print( " state ", i, " v =    ", t[i],
          " new policy =    ", test_1[i] ) );

                  117964980
 state  1  v =    ---------  new policy =     1
                   4805981

 state  2  v =    0          new policy =     2


(D32)       DONE
(C33) CLOSEFILE();
```

Figure 4.7: Third and Final Part of MACSYMA Output

Chapter 5

The Mine and Kawai Dynamic Programming Algorithm

5.1 Introduction

As documented in Chapter 2 there are a multitude of different repair, replacement and inspection time optimization scenarios that can be modelled. In this chapter the implementation of an example from the literature is used as an introduction to some of the possibilities for different models.

The objective is to determine the optimized inspection plan for a system subject to failure or degradation such that the cost per unit time over an infinite horizon is minimized. No discounting is taken into account. The state of the system is only known through inspection and the inspections are perfect.

The system consists of a single component that can be described as being in one of several states. A failure of the component is immediately obvious and corrective repairs are performed. A transition diagram is illustrated in Figure 5.1.

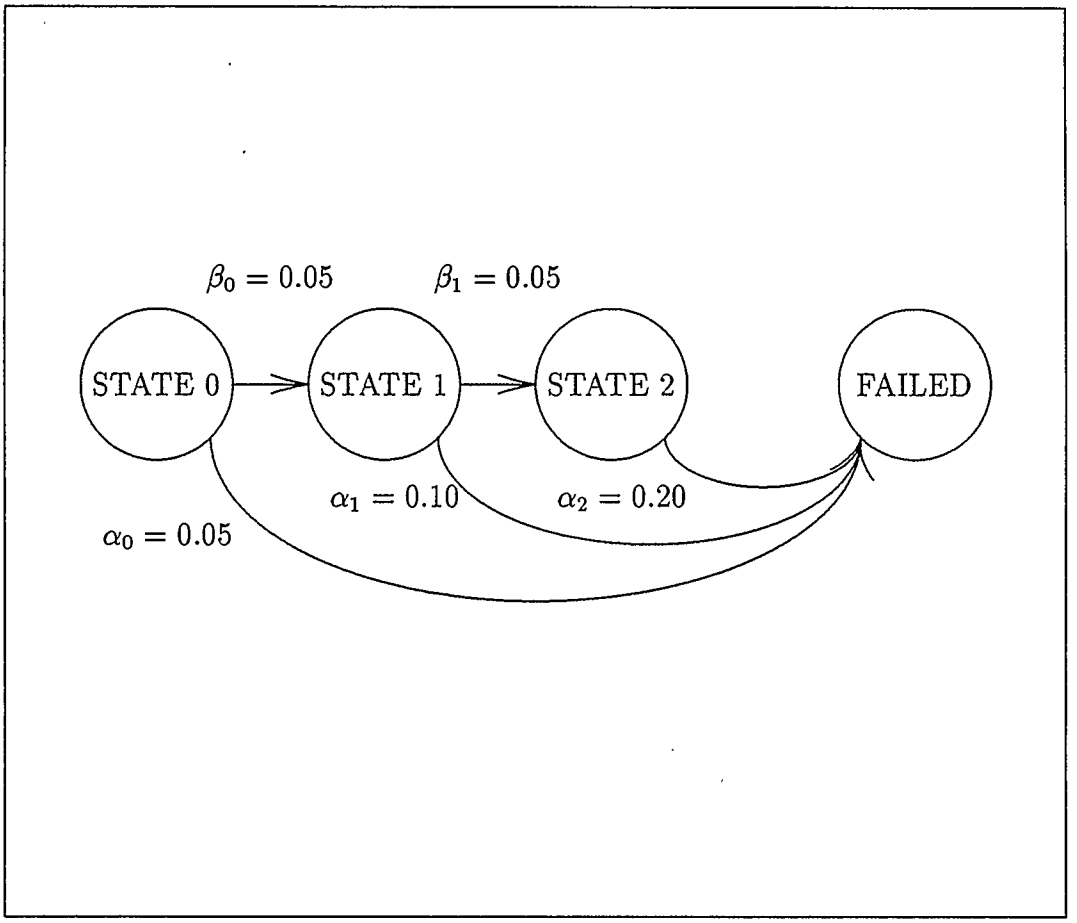


Figure 5.1: Transition Diagram for the Mine and Kawai Problem

5.2 The Mine and Kawai Model

5.2.1 Equations, Definitions and Notation

The system is modelled as a semi-Markov chain[1]. The time spent in each state is exponentially distributed. The system's states are numbered from 0 (perfect) to N (penultimate). State $N+1$ is the failed state. The only state obvious to the user is the failed state. Degradation, the gradual increase in state number from state i to state $i+1$, is modelled with parameter β_i. Failure, the immediate increase from state i to state $N+1$ (the failed state), is modelled with parameter α_i. In the Mine and Kawai model $\lambda_i = \alpha_i + \beta_i$.

Several of the other parameters are defined in Figure 5.2.

Three options are available to the user once the results of an inspection are known. The options are:

1. prevent a future failure by replacing the system immediately, denoted M,
2. do not replace the system, and never inspect the system again, denoted $I(\infty)$, a special case of the next option, or
3. plan to inspect the system at some future time, denoted $I(t)$.

The probability the system will be in state j, as a function of time t, given that the system was in state i at time 0, is given by

$$P_{ij}(t) = \beta_i \beta_{i+1} \ldots \beta_{j-1} \sum_{k=i}^{j} \left\{ \frac{e^{-\lambda_k t}}{\prod_{l=i, l \neq k}^{j} (\lambda_l - \lambda_k)} \right\}, \text{ for } i < j.$$

[1]The authors refer to the model as a semi-Markov process in the paper reviewed in this chapter, and as a continuous-time Markov process in the paper reviewed in Chapter 6.

Description	*Definition*
probability unit is in state i at time t if it is in state i at time 0	$P_i(t) = e^{-\lambda_i t}, \left(\bar{P}_i(t) = 1 - P_i(t)\right)$
failure rate from state i to state $N+1$	α_i, $\alpha_i < \alpha_j$ for $i < j$
degradation rate from state i to state $i+1$	β_i, $(\beta_N = 0), \beta_i \leq \beta_j, i < j$
total degration and failure rate	$\lambda_i = \alpha_i + \beta_i$
inspection cost	c_I
repair cost per unit time	c_r
maintenance cost per unit time	c_p
cost rate or cost per unit time for a given policy (this is the parameter to be minimized)	g
relative value of a state	ν_i, ($\nu_0 = 0$ by definition)

Figure 5.2: Definitions of the Mine and Kawai Problem

The distribution function of the failure of the system is given by

$$F_i(t) = 1 - \bar{F}_i(t) = 1 - \sum_{j=i}^{N} P_{ij}(t).$$

The one step transition probability is given by

$$\text{one step trans. prob.} = \begin{cases} P_{ij}(t) \text{ to state } 1, \ldots, N & \text{if system has not failed, or} \\ F_i(t) \text{ to state } 0 & \text{if system has failed.} \end{cases}$$

The cost to the next transition is given by

$$C_i(t) = \begin{cases} c_r T_r F_i(t) + c_I \bar{F}_i(t) & \text{if } D_i = I(t), \\ c_p T_p & \text{if } D_i = M. \end{cases}$$

The time to the next transition is given by

$$T_i(t) = \begin{cases} \int_0^t \bar{F}_i(x)dx + T_r F_i(t) & \text{if } D_i = I(t), \\ T_p & \text{if } D_i = M. \end{cases}$$

Mine and Kawai use a shorthand notation in their dynamic programming routine.

$$H_i(t,g) = \left[C_i(t) - gT_i(t) + \sum_{j=i+1}^{N} P_{ij}(t)\nu_j(g) \right] / \bar{P}i(t).$$

The formula has the units of cost per unit time. Recall that the ν_j are the relative values or costs of a state. For a given state, the minimum of the cost rate for the inspection option is given by the above formula. The product of cost rate and time until the next inspection is subtracted from the cost per unit time until the next replacement of the system. To this sum is added the relative value of all the $j = i + 1$ to N states weighted by the probability the system moves to those states in time t.

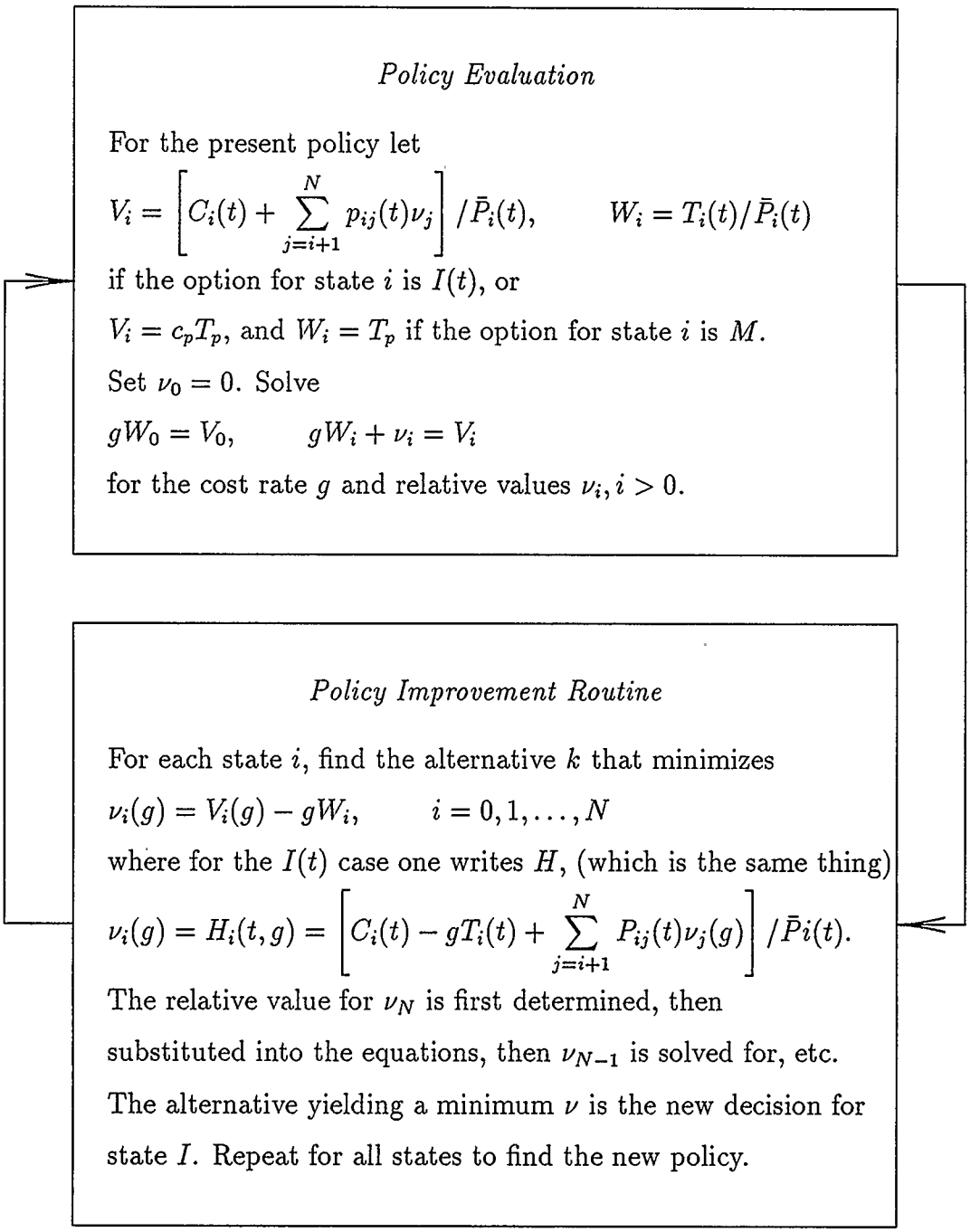


Figure 5.3: The Dynamic Programming Algorithm for the Mine and Kawai Problem

5.2.2 The Dynamic Programming Algorithm

A flow chart for the algorithm is illustrated in Figure 5.3.

When the option is to inspect at some later time (including $t = \infty$), the equations are as follows:

$$\begin{aligned} V_i &= [C_i(t) + \sum_{j=i+1}^{n} P_{ij}(t)\nu_j]/\bar{P}_i(t) \\ W_i &= T_i(t)/\bar{P}_i(t). \end{aligned}$$

The maintenance option equations are

$$\begin{aligned} V_i &= c_p T_p \\ W_i &= T_p. \end{aligned}$$

The following set of equations are solved for g and all the ν's but ν_0, which is arbitrarily set equal to zero. The relative values are determined if ν_0 is set to 0.

$$\begin{aligned} gW_0 &= V_0 \\ gW_i + \nu_i &= V_i, \text{ for } i \geq 1. \end{aligned}$$

If the g obtained from solving the system of equations is the same as the previous iteration the optimal solution has been found. If g is different the policy improvement routine (PIR) is initiated.

The details of the algorithm are as follows. To begin, pick an arbitrary decision or alternative for each state as the initial policy. The easiest starting option is to assume that, no matter what state the system is inspected to be in, maintain the system immediately. This choice results in the least amount of effort in getting through the first step.

The policy evaluation routine consists of three steps:

1. *The Policy Evaluation Routine* (PE): For the policy create the equations in V, W and ν.

2. Solve the equations for g and $\nu_1, \ldots, \nu_N$.

3. If g obtained from this set of equations is the same as obtained for the last iteration then the optimal solution has been found; otherwise implement the policy improvement routine (PIR).

The policy improvement routine consists of three steps:

1. *The Policy Improvement Routine* (PIR): Beginning with state N and working backwards to state 0, determine the decision or alternative that minimizes the relative cost ν_j of the current state. Note that for state N it can be shown that the only viable alternatives are M and $I(\infty)$. Use the alternative that gives the minimum in all further calculations in the PIR. The objective is to create a set of alternatives that minimizes the relative value of ν_0.

2. *How to get time into the problem*: H has only one minimum. The minimum may occur at ∞. If the minimum occurs for $t < \infty$ then this t must be determined. The minimum can be found by differentiating $H_i(t, g)$ and solving for H. MACSYMA (surprisingly!) does not have a function to determine where the minimum of a function occurs. To determine were the minimum occurs, H is differentiated with respect to time, and the zero of dH/dt is determined using the MACSYMA routine `root_by_bisection`. The ν_i is determined as $\min\{H_i(t^*(g), g), c_p T_p - g T_p\}$, where t^* is the time of the minimum.

3. Return to the policy evaluation (PE) routine to check the new solution.

5.3 Comparison of Results

Mine and Kawai give results for selected values of the parameters. For a unit with two degraded states, four types of policy are possible, viz. $[M, M, M]$, $[I(t_o), M, M]$, $[I(t_0), I(t_1), M]$ and $[I(\infty), I(\infty), I(\infty)]$.

The values of their parameters are $\lambda_0 = 0.10$, $\lambda_1 = 0.15$, and $\lambda_2 = 0.20$, which means $\alpha_2 = 0.20$ since $\beta_2 = 0.0$. Continuing, $\alpha_0 = 0.05$, $\alpha_1 = 0.10$, $c_r = 10$, $T_r = 4$, $T_p = 2$ and $c_I = 1$. The maintenance cost c_p is given the values 2, 5, 10 and 13 to illustrate the different results possible with the model.

To illustrate again how time enters into the problem, consider the State 0 case of the $c_p = 5$ example on the first iteration. The output from the program for H is

$$H = \frac{-2625e^{t/10} + 4324e^{t/20} - 1481}{218e^{t/5} - 218e^{t/10}}$$

Obviously $I(\infty) = 0$ because the exponent of the leading exponential in the denominator is greater than those in the numerator. The costs and times are such that $M = 5.60$. In Figure 5.4 an illustration of H, $I(\infty)$ and M illustrates that the minimum of H is lower than M and that the minimum occurs at 8.82 years, and has a relative value of -1.478. Thus, of the three possible decisions, the decision selected would be to do nothing now and reinspect the system 8.82 years in the future.

It was expected that the implementation of the Mine and Kawai dynamic programming algorithm for optimized inspection times would be simple, and would also serve as a good stepping stone to solving more complicated models. This did not turn out to be so. Fully six weeks of full–time work on the implementations was not able to create results equal to those of their paper. A comparison is given in Figure 5.5.

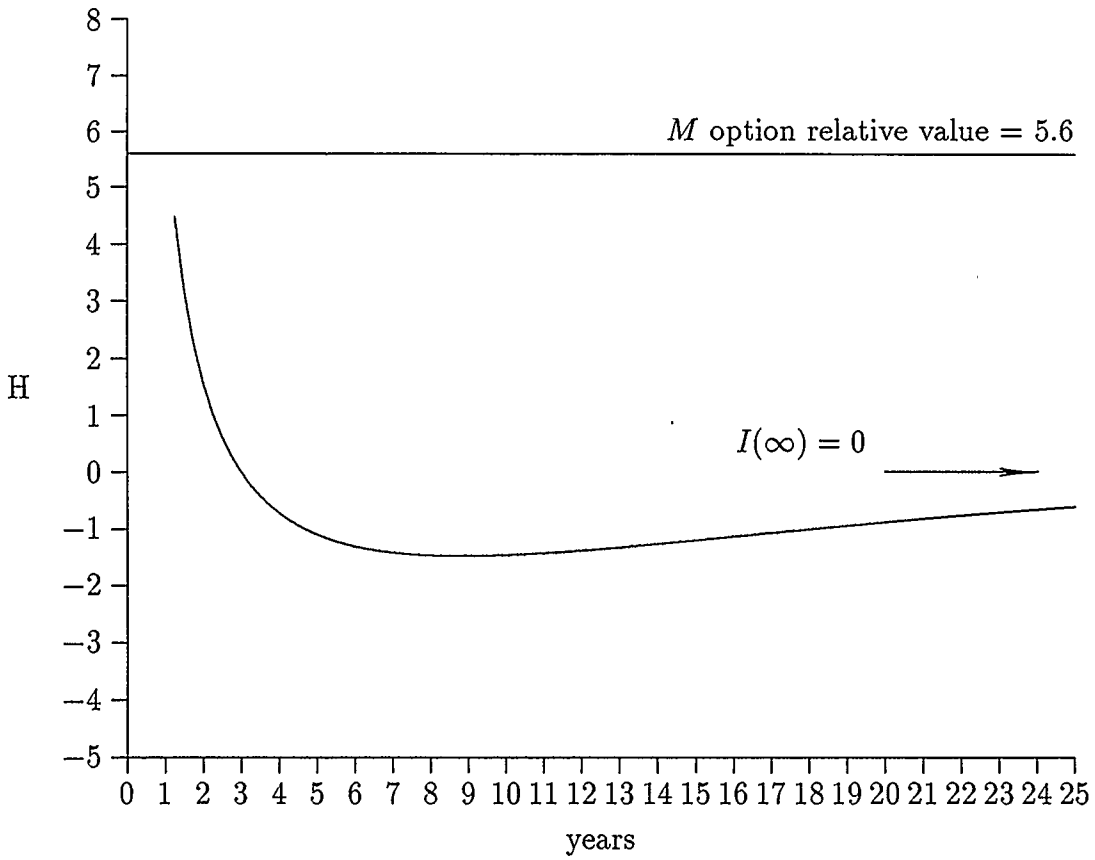


Figure 5.4: Illustration of Minimum of H in Comparison to Replace Option

c_p	*Optimal Policy*	*Cost Rate g*
2	$[M, M, M]$	2.00
5	$[15.10, M, M]$	2.15
10	$[34.30, 11.24, M]$	2.18
13	$[I(\infty), I(\infty), I(\infty)]$	2.20

Mine and Kawai Inspection Policy

c_p	*Optimal Policy*	*Cost Rate g*
2	$[M, M, M]$	2.00
5	$[8.16, M, M]$	2.10
10	$[I(\infty), 11.36, M]$	2.20
13	$[I(\infty), I(\infty), I(\infty)]$	2.20

Inspection Policy for this Project

Figure 5.5: Comparison of Inspection Policies for the Mine and Kawai Work, and for this Project

Note that for the $c_p = 10$ case the work for this project does not appear to be among the valid selections from the Mine and Kawai paper, although the results are reasonable. This is most frustrating. A more detailed comparison, for intermediate values of the maintenance cost, is given in Figures 5.6 and 5.7.

The only reasonable explanation for the lack of a match is that somehow the author has not implemented the correct algorithm. Several attempts over a six week period were made to compare the software and the algorithm. After minor corrections no difference could be found between the software implemented for this project and the algorithm as given in the Mine and Kawai paper.

5.4 A Full Optimized Inspection Analysis

A full optimized inspection analysis was conducted. The system modelled was a three state system, the states being numbered 0, 1 and 2 in order of increasing deterioration. The values of the parameters are shown in Figure 5.8.

The major portion of the effort of the analysis is determining whether the function H has a minimum, and whether the minimum value of H (i.e. the cost rate for inspecting t or ∞ years in the future) is less than the cost rate calculated for preventive maintenance of the system (i.e. immediate replacement).

In State 2 the function H does not have a minimum. The only available options are to replace the system immediately or let the system continue to decay for an infinite amount of time. This situation is unique to the state penultimate to failure of the system. In the other states of lesser deterioration the function H may have a minimum. The results for State 1 are shown in Figure 5.9 and the results for

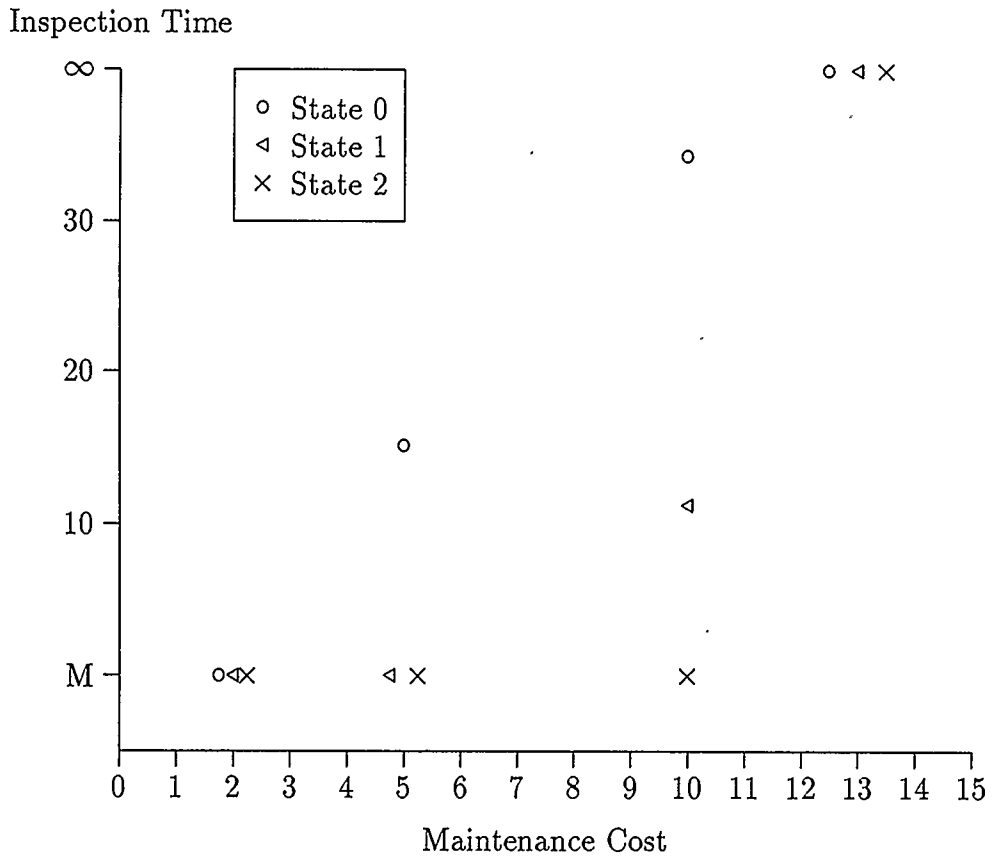


Figure 5.6: Mine and Kawai Inspection Policy

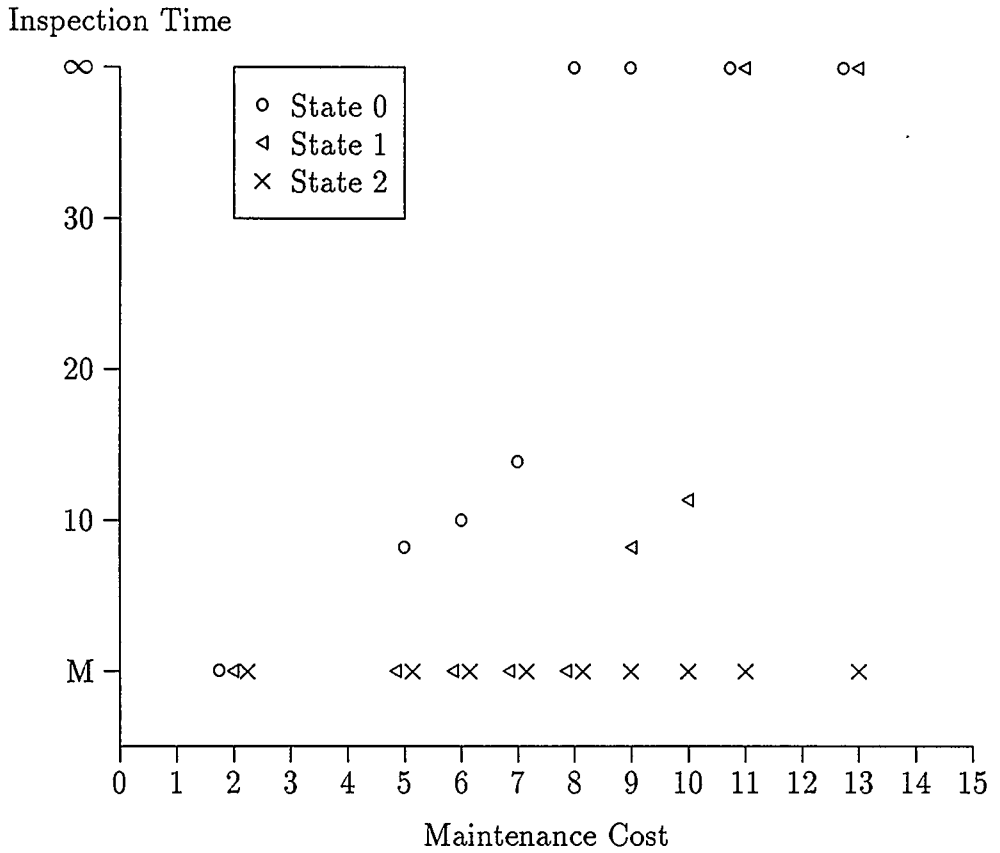


Figure 5.7: Inspection Policy for this Project

parameter	Value
cost of repair c_r	15
time of repair t_r	4
time of maintain t_p	2
cost of inspection c_I	1
cost of maintain c_p	10

Figure 5.8: Values Used in the Example

State 0 are shown in Figure 5.10. In both cases the preventive maintenance option M is also illustrated. In both figures the solid line is the second iteration and the dashed line represents the third and subsequent iterations. The convergence to the inspection time occurs very quickly. The first evaluation of H is off the figures in both cases. In the State 1 case the preventive maintenance option has a lesser cost than inspecting approximately four years in the future: the optimal decision when the system is inspected to be in State 1 is to maintain the system immediately, returning the system to State 0. In the State 0 case the optimized inspection time of approximately 9 years has a cost rate far less than that for preventive maintenance: the optimum decision when the system is observed to be in State 0 is to inspect the system 9 years in the future.

One advantage of using a computer algebra system is that the function H can be determined in a the form of a formula instead of a list of numbers. To illustrate some of the flavour of the output the formulas generated for each iteration are listed below.

$$H_{1,\text{first iteration}} = -\frac{190\,e^{t/5} - 196\,e^{t/20} + 3}{3\,e^{t/5} - 3\,e^{t/20}}$$

$$H_{1,\text{second iteration}} = \frac{2100\,e^{t/5} - 3722\,e^{t/20} + 1731}{109\,e^{t/5} - 109\,e^{t/20}}$$

$$H_{1,\text{third iteration}} = 22660861599517428620\,e^{t/5} - 39935631019167555306\,e^{t/20} + 18393182946617172483 \Big/ 1118413526967045797\,e^{t/5} - 1118413526967045797\,e^{t/20}$$

$$H_{0,\text{first iteration}} = -\frac{730\,e^{t/5} - 745\,e^{t/10} + 12\,e^{t/20} - 3}{6\,e^{t/5} - 6\,e^{t/10}}$$

$$H_{0,\text{second iteration}} = -\frac{2575\,e^{t/10} - 4524\,e^{t/20} + 1731}{218\,e^{t/5} - 218\,e^{t/10}}$$

$$\begin{aligned} H_{0,\text{third iteration}} = {} & 3280148857205501240\,e^{t/5} - 3219911387636329509 5\,e^{t/10} \\ & +4954897501970905793 2\,e^{t/20} - 1839318294661717248 3 \Big/ \\ & 2236827053934091594\,e^{t/5} - 2236827053934091594\,e^{t/10} \end{aligned}$$

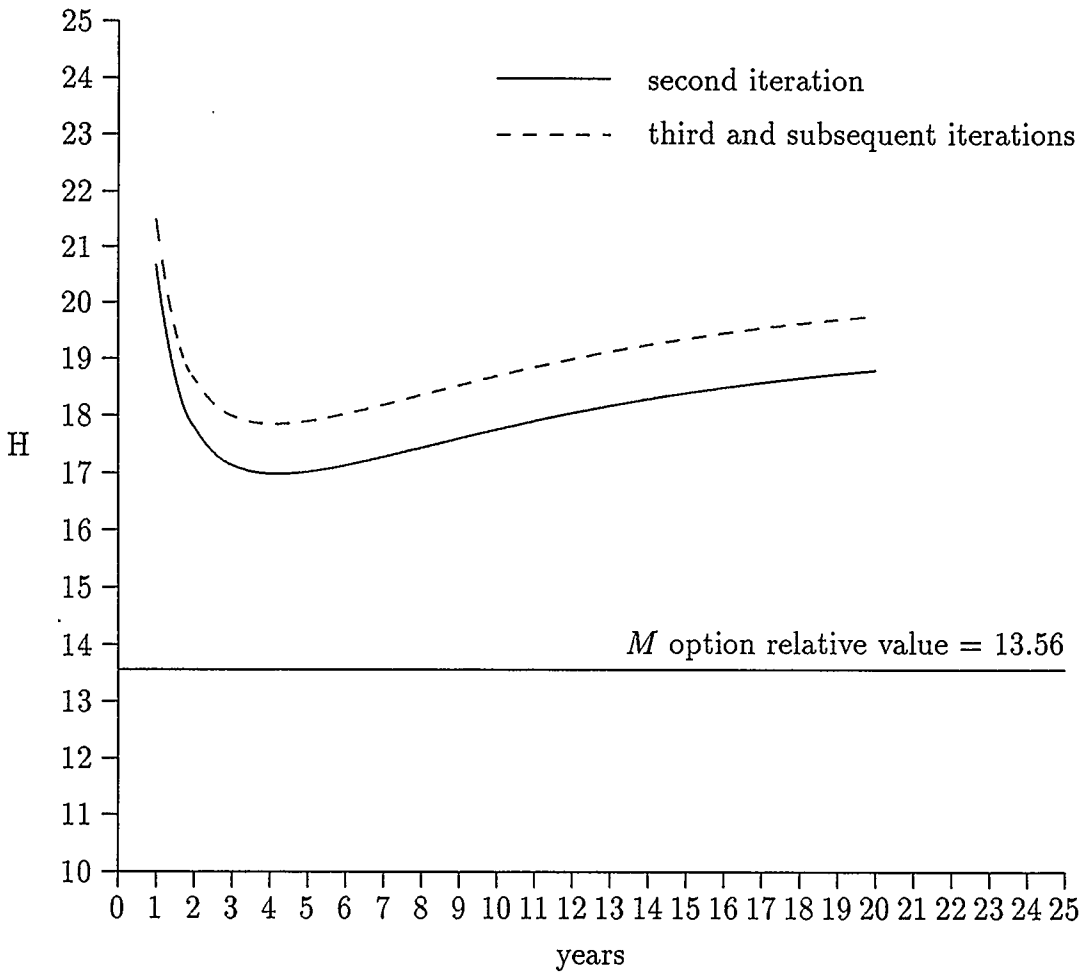


Figure 5.9: Minimum of H in Comparison to Replace Option for State 1

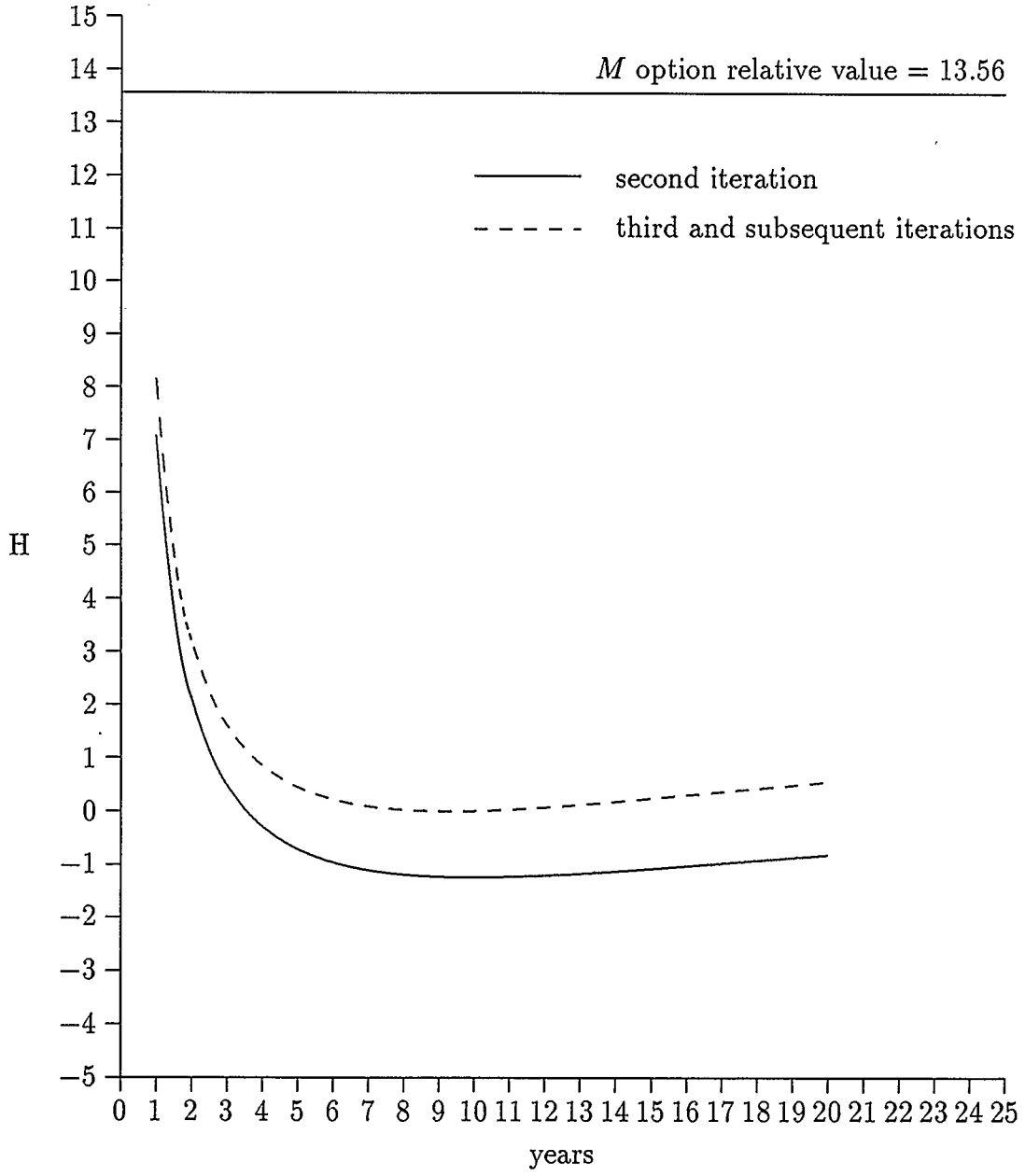


Figure 5.10: Minimum of H in Comparison to Replace Option for State 0

Chapter 6

The Ohnishi, Kawai and Mine Dynamic Programming Algorithm

6.1 Introduction

In contrast to the model presented in the previous chapter, in this chapter the results of implementing a model incorporating operating costs for a given state are presented. Ohnishi, Kawai and Mine [40] minimize the average cost per unit time. They use successive approximations to determine optimized inspection times.

The successive approximation methodology is implemented as:

1. Begin with an arbitrary policy to initialize the model (the simplest policy is one where for each state one chooses the option to replace immediately), as was the case for the model examined in the previous chapter.

2. Determine, for the initial set of options, the cost per unit time.

3. Perform the successive approximation analysis,

   - beginning at the penultimate state, determine the optimized strategy, i.e. select one of: i) replace immediately, ii) let the system fail naturally and wait until failure to replace, or iii) determine whether it is most cost effective to inspect the system t years in the future.

4. Determine, for the new policy, whether the cost per unit time is the same as that of the previous step. If the costs are the same then the optimal policy has been determined; if not, then perform the successive approximation algorithm again.

The model incorporates the same variables as those given in the previous chapter except for a state dependent operating cost a_i[1] and a state dependent replacement cost C_i. The transition scheme is identical to that of the Mine and Kawai model presented in Chapter 5 (see Table 5.2), and most importantly,

$$\lambda_0 \leq \lambda_1 \leq \ldots \leq \lambda_N,$$

and

$$\frac{\alpha_0}{\lambda_0} \leq \frac{\alpha_1}{\lambda_1} \leq \ldots \leq \frac{\alpha_N}{\lambda_N}(= 1).$$

What these two sets of equations mean is that both the degradation and failure rate increase with increasing system degradation. The system can only decay more quickly as a function of increasing decay.

The costs of the system have to satisfy stringent criteria as well.

$$C_0 \leq C_1 \leq \ldots \leq C_N \leq C_{N+1},$$

and

$$\frac{a_0}{\lambda_0} - C_0 \leq \frac{a_1}{\lambda_1} - C_1 \leq \ldots \leq \frac{a_N}{\lambda_N} - C_N.$$

The system becomes more costly to operate as it deteriorates. The system becomes more costly to replace as it deteriorates. Replacement becomes a better option with increasing deterioration.

[1]Note, α_i = failure rate (transition rate from state i to state N), and a_i = cost per unit time to operate in state i.

Failure of the system is obvious to the user, and a failed system is immediately replaced.

The probability transition scheme, coupled with the costs, is necessary for the function G (same idea as given by H in the previous chapter) to have a minimum. The development of the Ohnishi, Kawai and Mine model is now considered.

6.2 The Model

The failure probability as a function of time is

$$F(t) \equiv P_{i,N+1}(t)$$

where $P_{i,j}(t)$ is the probability the system is in state j at time t, given that it was in state i at time 0 (recalling that in all cases it is assumed that the transition parameters do not change with time).

$$F_i(t) \equiv 1 - \bar{F}_i(t) = 1 - \sum_{j=i}^{N} P_{ij}(t).$$

Another function that has to be defined is

$$Q_{ij}(t) \equiv \int_0^t P_{ij}(u)du.$$

The policy is denoted as δ, i.e. δ is a vector of decisions. If the optimum policy is to inspect 5 years in the future if the system is observed in state 0, and to maintain the system immediately if the system is observed in either of states 1 or 2, then $\delta = [5, M, M]$.

The function G is evaluated based on the time and costs to the next replacement of the system. The expected time from an inspection to the next *replacement* is

given by

$$x_\delta(i) = \begin{cases} \int_0^t \bar{F}_i(u)du + \sum_{j=i}^{N+1} P_{ij}(t)x_\delta(j) & \text{if } D_\delta(i) = I(t) \\ \int_0^\infty \bar{F}_i(u)du & \text{if } D_\delta(i) = I(\infty) \\ 0 & \text{if } D_\delta(i) = M. \end{cases}$$

The cost is made up of several components. In the $I(t)$ option, to inspect the system t years in the future, the time is the expected value of $\bar{F}$ and the times for all states $j = i$ to $N + 1$ weighted by the probability the system enters those states. If the time of next inspection is infinite the expected time until the next replacement is simply the expected value of $\bar{F}$. If the system is to be replaced in state i the time until the next replacement is zero.

The expected cost from an inspection to the next replacement is given by

$$y_\delta(i) = \begin{cases} M\bar{F}_i(t) + \sum_{j=i}^{N} \int_0^t P_{ij}(u)du\, a_j + \sum_{j=i}^{N+1} P_{ij}(t)y_\delta(j) & \text{if } D_\delta(i) = I(t) \\ \sum_{j=i}^{N} \int_0^\infty P_{ij}(u)du\, a_j + C_{N+1} & \text{if } D_\delta(i) = I(\infty) \\ C_i & \text{if } D_\delta(i) = M. \end{cases}$$

The cost until the next replacement formula is similar to the time formula. If the option is to inspect t years in the future, the cost to the next replacement is the sum of the inspection cost, weighted by $\bar{F}$, the cost per unit time of being in each state i to N weighted by the probability of entering the states and the weighted cost of all states of i to $N + 1$. If the option is to inspect the system at infinite time in the future the inspection cost M drops out and the cost is the sum of the state dependent costs per unit time and the cost of replacing the failed system (C_{N+1}). If the option is to replace the system immediately the cost is that of replacing the system in its current state (C_i).

During the successive approximation cycle, g is determined by setting

$$g = \frac{y_\delta(0)}{x_\delta(0)}.$$

The model is set up such that g is approached monotonically from above.

The function (similar to H in the previous chapter) is given by

$$\begin{aligned} G_i(t;g,\delta) \equiv & \frac{1}{1-P_{ii}(t)} \left\{ M\bar{F}_i(t) + \sum_{j=i}^{N} \int_0^t P_{ij}(u)du\, a_j \right. \\ & \left. + \sum_{j=i}^{N+1} P_{ij}(t)\left[y_\delta(j) - g x_\delta(j)\right] - \int_0^t \bar{F}_i(u)du \right\}. \end{aligned}$$

The denominator of the quotient goes to zero as t goes to zero. The limits of G as time approaches 0 and ∞ are:

$$\lim_{t \downarrow 0} G_i(t;g,\delta) = \infty,$$

and

$$\lim_{t \to \infty} G_i(t;g,\delta) = \sum_{j=i}^{N} \int_0^\infty P_{ij}(u)du\, a_j + C_{N+1} - g\int_0^\infty \bar{F}_i(u)du.$$

A flow chart for the Ohnishi, Kawai and Mine algorithm is given in Figure 6.1.

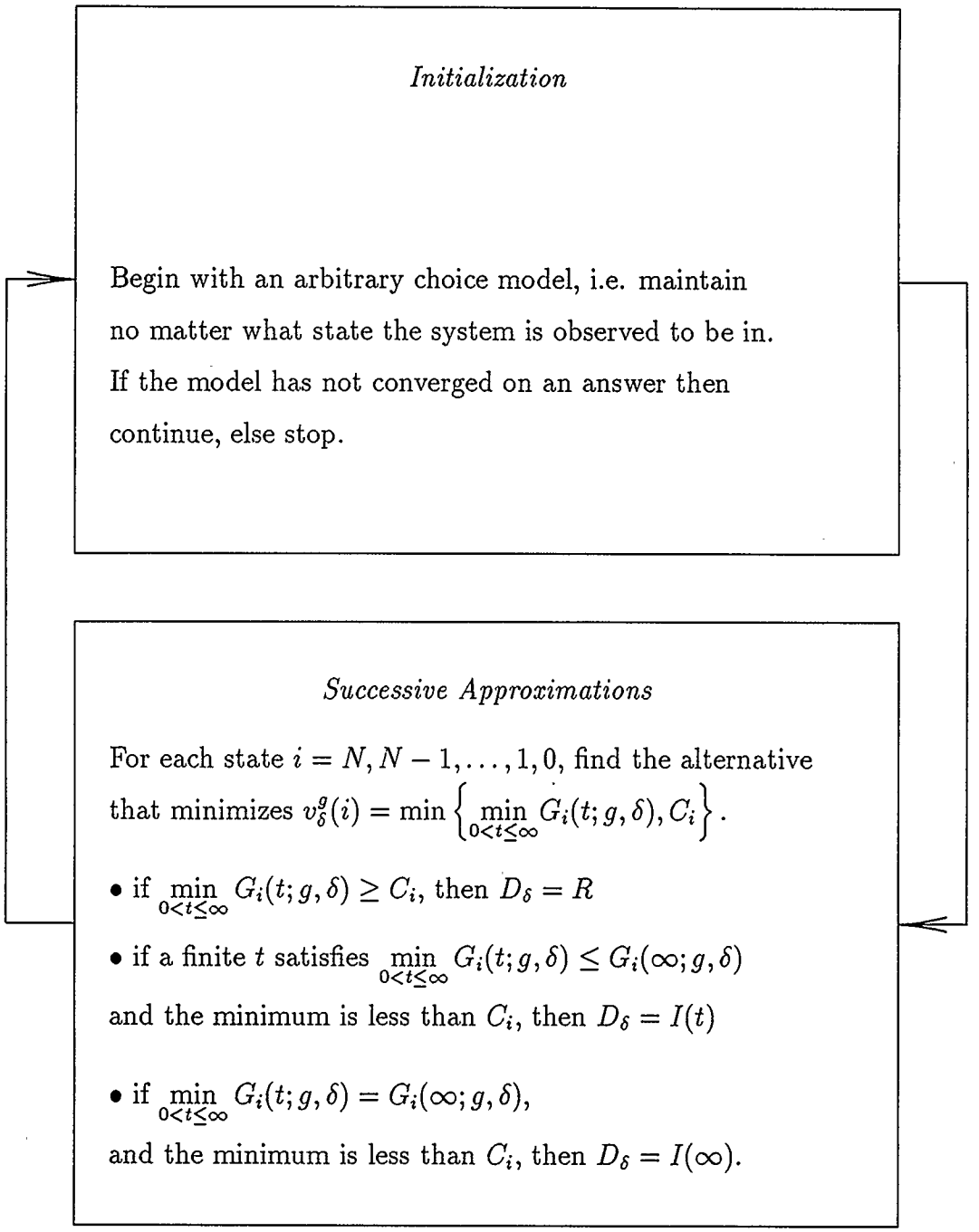


Figure 6.1: Dynamic Programming Algorithm: Ohnishi, Kawai and Mine Problem

6.3 A Full Optimized Inspection Analysis

The values for G_i for states 2, 1 and 0 are given in Figure 6.3, Figure 6.4 and Figure 6.5. The values used in the analysis are shown in Figure 6.2.

Parameter	Value
operating costs	[5, 10, 15, 20]
replacement costs	[10, 15, 20, 30]
time of maintain t_p	2
inspection cost	10
λ_i	[0.1, 0.15, 0.2]
β_i	[0.05, 0.05, 0]

Figure 6.2: Values Used in the Example

The formula for G becomes more complicated with lower state number and increasing number of iterations.

6.3.1 State 2 Equations

$$G_{2,\text{1st iteration}} = \frac{70\,e^{t/5} - 60}{e^{t/5} - 1}$$

$$G_{2,\text{2nd iteration}} = \frac{25\,e^{2t/5} + 10\,e^{t/5} - 24}{e^{2t/5} - e^{t/5}}$$

$$G_{2,\text{3rd iteration}} = \frac{31335\,e^{3t/5} + 7610\,e^{2t/5} + 7610\,e^{t/5} - 28112}{761\,e^{3x/5} - 761\,e^{2x/5}}$$

6.3.2 State 1 Equations

$$G_{1,\text{1st iteration}} = \frac{-250\,e^{2t/5} + 190\,e^{t/4} + 30\,e^{t/5} + 180\,e^{t/20} - 180}{3\,e^{t/4} - 3\,e^{2t/5}}$$

$$G_{1,\text{2nd iteration}} = \frac{25\,e^{3t/5} + 60\,e^{9t/20} - 30\,e^{2t/5} + 8\,e^{3t/10} - 30\,e^{t/5} - 72\,e^{t/10} + 72}{3\,e^{3t/5} - 3\,e^{9t/20}}$$

$$G_{1,\text{3rd iteration}} = 80575\,e^{4t/5} + 45660\,e^{13t/20} - 22830\,e^{3t/5} + 45660\,e^{t/2} - 22830\,e^{2t/5} - 48076\,e^{7t/20} - 22830\,e^{t/5} - 84336\,e^{3t/20} + 84336 \Big/ 2283\,e^{4t/5} - 2283\,e^{13t/20}$$

6.3.3 State 0 Equations

$$G_{0,\text{1st iteration}} = 245\,e^{11t/20} - 170\,e^{9t/20} - 60\,e^{2t/5} + 15\,e^{7t/20} - 190\,e^{3t/10} + 70\,e^{t/4} + 210\,e^{t/5} - 90\,e^{3t/20} - 180\,e^{t/10} + 360\,e^{t/20} - 180 \Big/ 3\,e^{11t/20} - 3\,e^{9t/20}$$

$$G_{0,\text{2nd iteration}} = -\Big(275\,e^{3t/4} - 150\,e^{13t/20} + 120\,e^{3t/5} - 401\,e^{11t/20} + 120\,e^{9t/20} - 16\,e^{2t/5} + 102\,e^{7t/20} - 188\,e^{3t/10} + 204\,e^{t/4} - 60\,e^{t/5} - 72\,e^{3t/20} - 144\,e^{t/20} + 144\Big) \Big/ 6\,e^{3t/4} - 6\,e^{13t/20}$$

$$G_{0,\text{3rd iteration}} = -5e^{19t/20} - 114150e^{17t/20} + 91320e^{4t/5} - 136980e^{3t/4} + 3487e^{13t/20} - 22830e^{11t/20} + 187472e^{t/2} + 129996e^{9t/20} - 214332e^{2t/5} + 95350e^{7t/20} - 45660e^{3t/20} + 45660e^{t/4} - 45660e^{t/5} - 253008e^{3t/20} + 168672e^{t/10} - 168672e^{t/20} + 168672 \Big/ 4566e^{19t/20} - 4566e^{t/20}$$

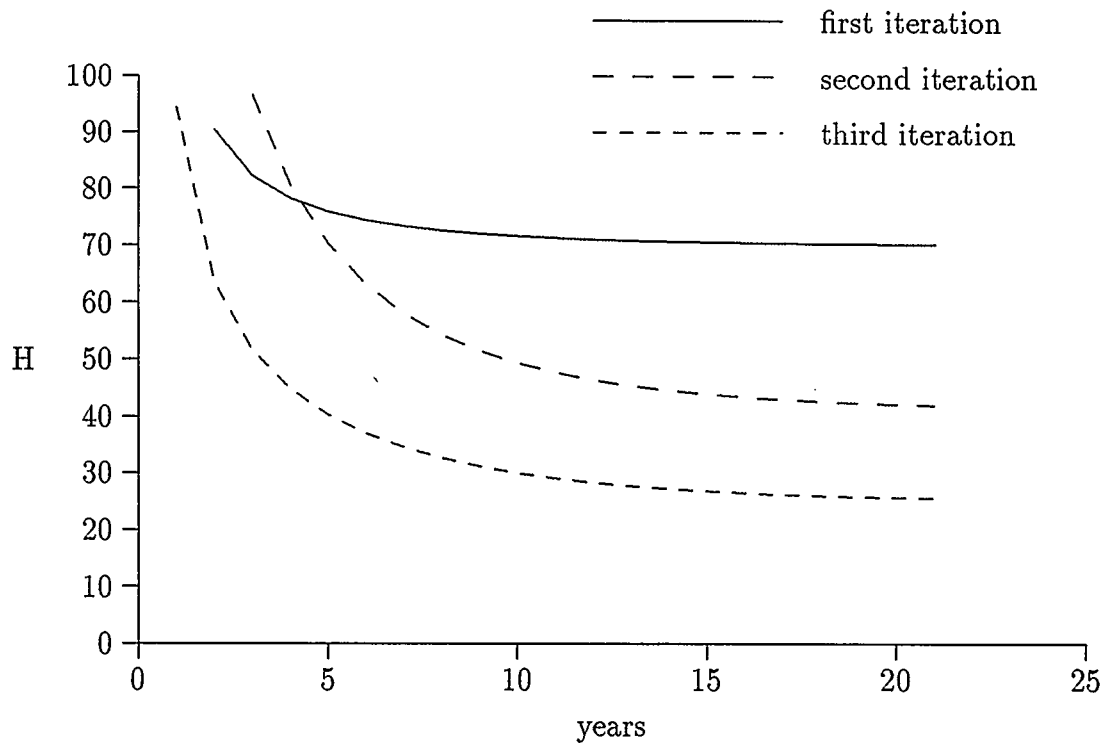


Figure 6.3: Illustration of G for State 2

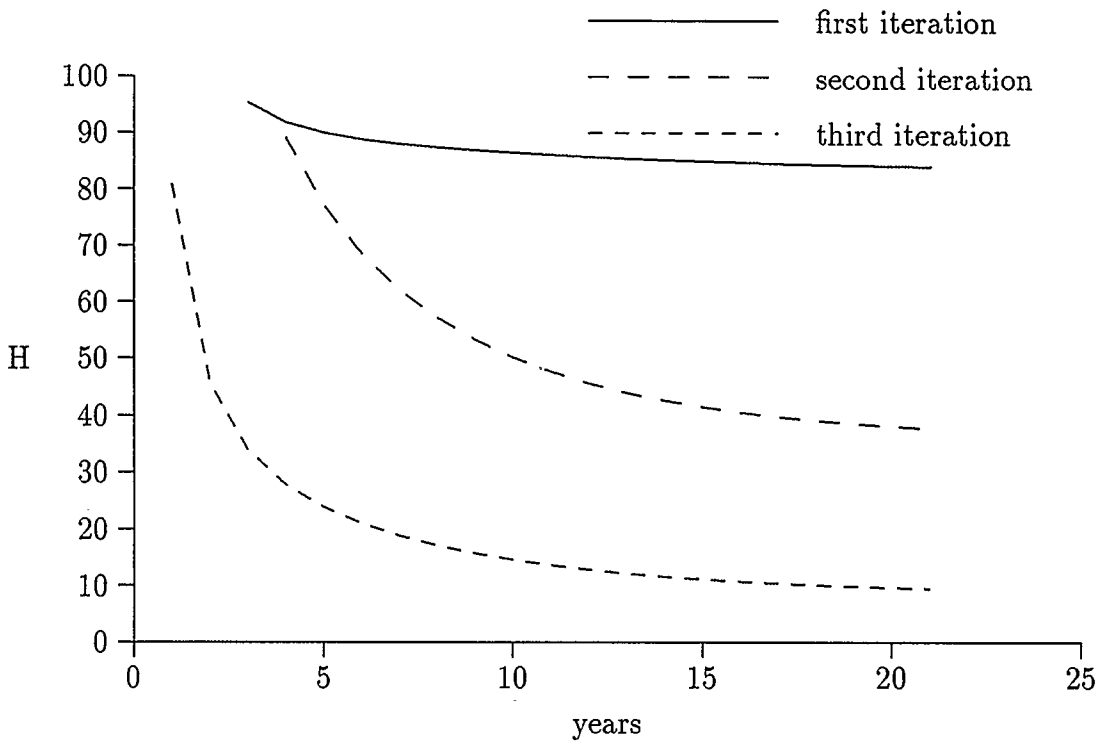


Figure 6.4: Illustration of G for State 1

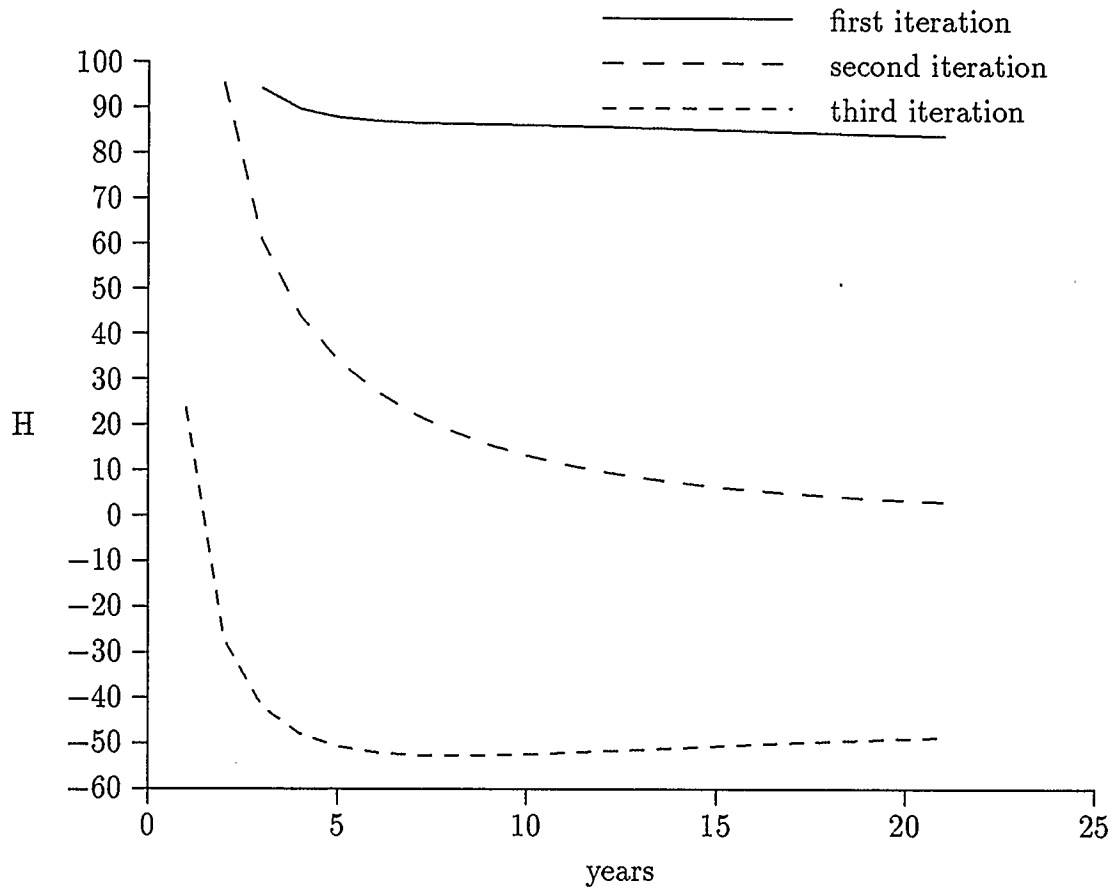


Figure 6.5: Illustration of G for State 0

Chapter 7

The Real World

7.1 Introduction

This chapter was prepared for a project Morrison Scientific Problem Solving is conducting for NOVA Corporation of Alberta. This chapter is included in this thesis because the discussion herein is important for determining the state specific transition probabilities that are to be incorporated into models such as those presented in Chapters 3, 5 and 6. The Mine group models assume very specific transition parameters. In all of their models the system can only pass from one state to the next state of higher severity or the system can fail. **The main question is**: Given a data set from the real world, is it possible to mould the real world data into a form amenable for analysis using the models presented earlier?

The best analysis would be for the data to be directly incorporated into one of the models presented in Chapters 5 or 6. An acceptable analysis, though not immediately capable of being examined using dynamic programming techniques, would be to mould the data into a form similar to that of the model presented in Chapter 3. In this latter form the reliability of the system as a function of time could be examined.

As a secondary goal, it is desired to know if the discrete data obtained on two separate inspections of a system can be incorporated into a system of differential equations. This is important because the matrix of transition probabilities is ob-

tained at discrete intervals. If the data can be made into a continuous form then the researcher would not have to worry about interpolating between discrete time periods.

In a rather cryptic comment Howard [10] mentions that to transform a discrete transition matrix into a continuous transition matrix that has the same state probabilities at multiples of the discrete time period it is necessary to solve

$$\mathbf{C} = \ln \mathbf{D},$$

where $\mathbf{C}$ is the continuous transition matrix and $\mathbf{D}$ is the discrete transition matrix. The cryptic comment is that "Methods for accomplishing this exist, ...", and the reader is referred to a 1959 MIT Operations Research Center Report. Paolucci [44] indicated how to determine the logarithm of a matrix.

Calculating the logarithm of a matrix that has multiple eigenvalues is not a trivial matter. For large matrices (e.g. 25×25, or even 10×10) the calculations necessary to calculate the logarithm of a matrix are prohibitive. One reason for the prohibitive nature of the calculations is roundoff errors when an infinite series approximation is used. The roundoff errors get out of hand for matrices on the order of 8×8. Another reason for the prohibitive nature of the calculations is that, to determine the logarithm of a matrix using the spectral resolution of $f(\mathbf{A})$, the function of a matrix $\mathbf{A}$, the set of linear equations that must be solved varies as the cube of the size n of the matrix. For an 8×8 matrix a system of 512 equations must be solved.

The studies presented in this chapter are also important in their own right because the connection between the discrete (matrix) analysis of a Markov chain and the continuous (differential equation) analysis is never examined deeply enough for the

reader to be able to answer the question: How are the elements of a Markov chain related to the coefficients of the differential equations?

First the raw data will be reviewed. Then the calculation of the logarithm using the different methods will be reviewed.

7.2 Determining a Transition Matrix from Observations: The Raw Data

Consider a decaying system. In this section a methodology for analyzing the decay of the system will be presented. In order to make the situation more real, the system will be a pipeline, and the data will be that data obtained from two different inspections of the pipeline.

Inspections are conducted according to some plan created by the user of the pipeline. During an inspection an in-line-inspection tool (or ILI tool) is passed through the pipeline. Transducers on the tool can indicate the characteristics of corrosion pits on the *outside* of the pipeline. It is possible to match corrosion pits from one inspection to a second inspection. What results is a matrix of frequencies of growth (on some scale determined by the researcher) during the time between the two inspections. New pits initiated between the inspections can be ignored in this analysis. It remains to transform the frequencies of growth into a useful model from which to garner information relating to the reliability of the pipeline as a function of time.

The form of the matrix that can be expected from this type of analysis is the following (an 8×8 example). Each value represents the frequency of observations of

a corrosion pit from an initial state to a state observed during the inspection.

$$\begin{pmatrix} 100 & 75 & 42 & 25 & 10 & 7 & 2 & 0 \\ 0 & 45 & 20 & 10 & 8 & 0 & 0 & 4 \\ 0 & 0 & 17 & 8 & 3 & 0 & 1 & 0 \\ 0 & 0 & 0 & 9 & 12 & 4 & 6 & 0 \\ 0 & 0 & 0 & 0 & 7 & 6 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

The matrix upper triangular because (ignoring possible resolution problems, etc.) the state of the system cannot decrease to one of lessor severity.

In terms of the risk analysis it is desired to replace all the zero's in the upper triangular portion of the matrix with non-zero values, otherwise a growing pit would have zero probability of entering certain states. Two options come to mind as to how to remove the zeros[1]

This is an inference problem. It now remains to determine a method whereby an estimate of all the values in the upper triangular portion of the matrix can be estimated. Among obvious candidates for estimating the values are

1. regression analysis, and

2. probability distribution analysis.

The regression analysis is problematic because of having to normalize the sum of the rows to 1.0. The probability distribution analysis has several advantages such as

1. ease of implementation,

2. several distributions can be used, such as

[1]In reality the matrix is of such size that only the upper leftmost corner of the matrix has non zero elements. Clearly, for a system operating far from failure, significant extrapolation is necessary.

- exponential,
- gamma,
- poisson,
- log-normal,

3. can be implemented with a minimal number of data points (a very important criterion, especially for higher severities where not many pits are observed),
4. rows are forced to sum to 1.0, and
5. error bars are easily estimated assuming binomial probabilities of an element of the matrix having that particular value determined by the probability distribution.

The only disadvantage of the probability distribution method is determining that distribution most appropriate for modelling a row of the matrix.

The easiest distribution to fit to the data is the exponential. When the cutoff of the distribution occurs at 0 the parameter for the exponential distribution is the average, therefore the value used to determine the exponential for a given row is the average change in state number between the inspections.

7.3 Making a Continuous Markov Model from a Discrete Model: The Logarithm of a Matrix

In this section two methods for determining the logarithm of a matrix will be reviewed[2].

What should the logarithm of the matrix look like? Based on the "data" matrix presented above one should see that transitions to higher states decrease as the higher state increases. As one examines a row of the logarithm of the matrix, one would expect to see decreasing values towards the right, except for possibly the rightmost element which is the sum from that state to ∞.

7.3.1 Logarithm of a Matrix: Infinite Series Approximation

The formula for the logarithm of a number is

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots, |x| < 1$$

Since the logarithm of x is desired, it can be determined by simply writing $x-1$ instead of x in the formula, giving

$$\ln(1+(x-1)) = (x-1) - \frac{(x-1)^2}{2} + \frac{(x-1)^3}{3} - \frac{(x-1)^4}{4} + \ldots, |x| < 1$$

The matrix **A** can be substituted for x in the above formula, yielding the continuous transition matrix. The identity matrix replaces "1."

[2]The methods reviewed here are not the only methods for determining the logarithm of a matrix. The presence of multiple eigenvalues significantly complicates the analysis. With time, as more of these methods become familiar to the author, a suitable method will be determined (or discovered). The important aspect of this chapter is the relationship between the discrete and continuous representations of the decay of the system.

Consider an original matrix (after fitting with the exponential distribution) given by

$$\begin{pmatrix} .181269 & .148411 & .670320 \\ 0 & .181269 & .818731 \\ 0 & 0 & 1 \end{pmatrix}.$$

After summing the first 10 elements of the series the estimate of the logarithm of the matrix is

$$\begin{pmatrix} -1.6653 & 0.70793 & 0.95737 \\ 0 & -1.6653 & 1.6653 \\ 0 & 0 & 0 \end{pmatrix}.$$

After 30 elements of the series the logarithm of the matrix is

$$\begin{pmatrix} -1.70745 & 0.8167 & 0.89075 \\ 0 & -1.70745 & 1.70745 \\ 0 & 0 & 0 \end{pmatrix}.$$

After 100 elements of the series the logarithm of the matrix is

$$\begin{pmatrix} -1.70776 & 0.8172 & 0.88903 \\ 0 & -1.70776 & 1.70776 \\ 0 & 0 & 0 \end{pmatrix}.$$

7.3.2 An 8×8 Example Using the Infinite Series Approximation

The calculation using the infinite series is fraught with roundoff errors and for large matrices the numerical instabilities quickly render the matrix unusable because many of the off-diagonal elements become negative.

A simulation was conducted. The original matrix used in the simulation was[3]

$$\begin{pmatrix}
3.2967b-1 & 2.2099b-1 & 1.4813b-1 & 9.9298b-2 & 6.656b-2 & 4.4616b-2 & 2.9907b-2 & 6.0809b-2 \\
0.0b0 & 5.5067b-1 & 1.4813b-1 & 9.9298b-2 & 6.656b-2 & 4.4616b-2 & 2.9907b-2 & 6.0809b-2 \\
0.0b0 & 0.0b0 & 2.4852b-1 & 1.8675b-1 & 1.4034b-1 & 1.0546b-1 & 7.9255b-2 & 2.3965b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 2.2119b-1 & 1.7226b-1 & 1.3416b-1 & 1.0448b-1 & 3.6787b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 1.9926b-1 & 1.5955b-1 & 1.2776b-1 & 5.1341b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 1.8126b-1 & 1.4841b-1 & 6.7031b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 1.8126b-1 & 8.1873b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 1.0b0
\end{pmatrix}.$$

All the rows were generated from exponential distributions. The first two rows were generated from the same exponential distribution to determine if the values in the logarithm of the matrix were the same.

The computer algebra system MACSYMA [58] was used to multiply the matrices using "bigfloats,", thus the exponents are b's instead of e's. The floating point precision (fpprec) parameter was set to 30 and 40 digits of precision and no differences were discerned in the final matrix.

After 30 elements of series the matrix was

$$\begin{pmatrix}
-1.1096b0 & 5.1301b-1 & 3.9005b-1 & 1.3815b-1 & 5.5772b-2 & 2.2011b-2 & 6.7985b-3 & -1.6177b-2 \\
0.0b0 & -5.9661b-1 & 3.9005b-1 & 1.3815b-1 & 5.5772b-2 & 2.2011b-2 & 6.7985b-3 & -1.6177b-2 \\
0.0b0 & 0.0b0 & -1.3922b0 & 7.9581b-1 & 3.0528b-1 & 1.4602b-1 & 7.8126b-2 & 6.6954b-2 \\
0.0b0 & 0.0b0 & 0.0b0 & -1.5086b0 & 8.1946b-1 & 3.2844b-1 & 1.6054b-1 & 2.0017b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.6129b0 & 8.3769b-1 & 3.3803b-1 & 4.3725b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.7074b0 & 8.167b-1 & 8.9074b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.7074b0 & 1.7074b0 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0
\end{pmatrix}.$$

[3]This matrix is not the exponential fit to the "raw" data matrix presented on page 72.

After 70 elements of series the matrix was

$$\begin{pmatrix}
-1.1096b0 & 5.1301b-1 & 3.9006b-1 & 1.3805b-1 & 5.6133b-2 & 2.1667b-2 & 6.6046b-3 & -1.5908b-2 \\
0.0b0 & -5.9661b-1 & 3.9006b-1 & 1.3805b-1 & 5.6133b-2 & 2.1667b-2 & 6.6046b-3 & -1.5908b-2 \\
0.0b0 & 0.0b0 & -1.3922b0 & 7.9608b-1 & 3.039b-1 & 1.485b-1 & 7.7466b-2 & 6.6253b-2 \\
0.0b0 & 0.0b0 & 0.0b0 & -1.5086b0 & 8.2017b-1 & 3.2559b-1 & 1.6407b-1 & 1.9883b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.6131b0 & 8.3922b-1 & 3.3386b-1 & 4.4003b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.7077b0 & 8.1873b-1 & 8.8904b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.7077b0 & 1.7077b0 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0
\end{pmatrix}.$$

After 100 elements of the series the matrix was

$$\begin{pmatrix}
-1.1096b0 & 5.1301b-1 & 3.9006b-1 & 1.3805b-1 & 5.6133b-2 & 2.1666b-2 & 6.6062b-3 & -1.5909b-2 \\
0.0b0 & -5.9661b-1 & 3.9006b-1 & 1.3805b-1 & 5.6133b-2 & 2.1666b-2 & 6.6062b-3 & -1.5909b-2 \\
0.0b0 & 0.0b0 & -1.3922b0 & 7.9608b-1 & 3.039b-1 & 1.485b-1 & 7.7459b-2 & 6.6259b-2 \\
0.0b0 & 0.0b0 & 0.0b0 & -1.5086b0 & 8.2017b-1 & 3.2559b-1 & 1.6408b-1 & 1.9883b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.6131b0 & 8.3922b-1 & 3.3386b-1 & 4.4003b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.7077b0 & 8.1873b-1 & 8.8903b-1 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & -1.7077b0 & 1.7077b0 \\
0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0 & 0.0b0
\end{pmatrix}.$$

Notice the two small negative values in the upper right hand corner of the matrix. These values should be positive. For matrices larger than 8×8 the number of negatives in the upper right hand corner grew rapidly, making these matrices all but useless.

The final matrix is the logarithm of the discrete matrix. The discrete matrix is used in the matrix multiplication method of determining probabilities. The continuous (logarithm) matrix is used in solving the problem using differential equations. The differential equations are shown in Figure 7.1.

The output from the differential equation (continuous) code is shown in Figure 7.2.

The comparison conducted above compared the discrete and continuous probabilities on an equal time basis. When the discrete data are obtained over a different time basis (such as four time periods, for example), the discrete and continuous

probabilities are compared by thinking "one time increment in the continuous case is equal to the number of time increments between measurements in the discrete case." Thus, for a four time period span between the measurements a "1" in the time column for the continuous calculation represents four periods of growth. To integrate the continuous equations one year at a time the user would integrate 0.25 time units at a time.

```
dydx[1]  = -1.1096e0  * y[1];
dydx[2]  =  5.1301e-1 * y[1] - 5.9661e-1 * y[2];
dydx[3]  =  3.9006e-1 * y[1] + 3.9006e-1 * y[2] - 1.3922e0  * y[3];
dydx[4]  =  1.3805e-1 * y[1] + 1.3805e-1 * y[2] + 7.9608e-1 * y[3] -
            1.5086e0  * y[4];
dydx[5]  =  5.6133e-2 * y[1] + 5.6133e-2 * y[2] + 3.0390e-1 * y[3] +
            8.2017e-1 * y[4] - 1.6131e0  * y[5];
dydx[6]  =  2.1666e-2 * y[1] + 2.1666e-2 * y[2] + 1.4850e-1 * y[3] +
            3.2559e-1 * y[4] + 8.3922e-1 * y[5] - 1.7077e0  * y[6];
dydx[7]  =  6.6062e-3 * y[1] + 6.6062e-3 * y[2] + 7.7459e-2 * y[3] +
            1.6408e-1 * y[4] + 3.3386e-1 * y[5] + 8.1873e-1 * y[6] -
            1.7077e0  * y[7];
dydx[8]  = -1.5909e-2 * y[1] - 1.5909e-2 * y[2] + 6.6259e-2 * y[3] +
            1.9883e-1 * y[4] + 4.4003e-1 * y[5] + 8.8903e-1 * y[6] +
            1.7077e0  * y[7];
```

Figure 7.1: The Continuous Time Differential Equations Determined From the Logarithm of the Discrete Transition Probability Matrix

time	y[1]	y[2]	y[3]	y[4]	y[5]	y[6]	y[7]	y[8]
0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1.00000	0.32969	0.22099	0.14814	0.09930	0.06656	0.04462	0.02991	0.06081
2.00000	0.10870	0.19455	0.11839	0.10432	0.08782	0.07223	0.05914	0.25489
3.00000	0.03584	0.13116	0.07435	0.07530	0.07227	0.06712	0.06201	0.48200
4.00000	0.01181	0.08014	0.04322	0.04713	0.04892	0.04909	0.04919	0.67054
5.00000	0.00390	0.04674	0.02436	0.02763	0.03005	0.03169	0.03356	0.80212
6.00000	0.00128	0.02660	0.01356	0.01569	0.01754	0.01908	0.02096	0.88535
7.00000	0.00042	0.01493	0.00750	0.00877	0.00996	0.01104	0.01242	0.93502
8.00000	0.00014	0.00832	0.00414	0.00487	0.00557	0.00624	0.00713	0.96365
9.00000	0.00005	0.00461	0.00228	0.00269	0.00309	0.00349	0.00402	0.97983
10.0000	0.00002	0.00255	0.00126	0.00148	0.00171	0.00193	0.00224	0.98887

Differential Equation Solution

time	y[1]	y[2]	y[3]	y[4]	y[5]	y[6]	y[7]	y[8]
0.00000	1.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
1.00000	0.32968	0.22099	0.14814	0.09930	0.06656	0.04462	0.02991	0.06081
2.00000	0.10869	0.19455	0.11839	0.10431	0.08781	0.07222	0.05913	0.25489
3.00000	0.03583	0.13115	0.07434	0.07529	0.07227	0.06711	0.06201	0.48199
4.00000	0.01181	0.08014	0.04321	0.04712	0.04892	0.04909	0.04919	0.67052
5.00000	0.00389	0.04674	0.02436	0.02762	0.03005	0.03169	0.03355	0.80209
6.00000	0.00128	0.02660	0.01356	0.01569	0.01754	0.01907	0.02095	0.88531
7.00000	4.233E-4	0.01493	0.00750	0.00877	0.00996	0.01103	0.01242	0.93497
8.00000	1.39553E-4	0.00832	0.00414	0.00487	0.00557	0.00624	0.00713	0.96360
9.00000	4.60081E-5	0.00461	0.00228	0.00269	0.00309	0.00349	0.00402	0.97977
10.0000	1.51679E-5	0.00255	0.00126	0.00148	0.00171	0.00193	0.00224	0.98881

Matrix Multiplication Solution

Figure 7.2: Comparison of Probabilities as a Function of Time from the Differential Equation and Matrix Multiplication Solutions

7.3.3 Logarithm of a Matrix Using the Spectral Resolution of f(A)

Lancaster and Tismenetsky [23] show how the spectral resolution of a function of a matrix can be used to determine the logarithm by solving a system of equations. This method is good for small matrices however the number of equations in the system varies as the cube of the size of the matrix. An 8×8 matrix would require 512 equations; a 25×25 matrix would require over 15,000 equations. A review of this method is included here for completeness.

The key of the calculation is that the function of a matrix can be written in the form

$$f(\mathbf{A}) = \sum_{k=1}^{s} \sum_{j=0}^{m_k-1} f_{k,j} Z_{kj},$$

where f is some polynomial and the Z's are matrices, linearly independent and capable of commuting with each other and the matrix $\mathbf{A}$. The spectrum of the matrix is the set of eigenvalues λ_k. The multiplicity of each eigenvalue is denoted m_k. The total number of different eigenvalues is s. The minimal polynomial is

$$\prod_k (\lambda - \lambda_k)^{m_k}.$$

A three by three example, with one repeated root, illustrates the technique well. Consider the following matrix

$$\begin{pmatrix} .181269 & .148411 & .670320 \\ 0 & .181269 & .818731 \\ 0 & 0 & 1 \end{pmatrix}.$$

The eigenvalues of this matrix are .181269, with $m_1 = 2$, i.e. this root is repeated, and 1, with $m_2 = 1$,

The minimal polynomial is

$$m(\lambda) = (\lambda - 1)(\lambda - .181269)^2.$$

The functions $f_{k,j}$ are linearly independent polynomials. In this case reasonable polynomials are $1, (\lambda - .181269)$, and $(\lambda - .181269)^2$. The function is given by

$$\begin{aligned} f(\mathbf{A}) &= \sum_{k=1}^{s} \sum_{j=0}^{m_k - 1} f_{k,j} Z_{kj}, \\ &= f(1)Z_{10} + f(.181269)Z_{20} + f'(.181269)Z_{21} \end{aligned}$$

The polynomials are substituted into the above equation and the Z's are determined from the resulting system of equations.

The $f(\lambda) = 1$ Equation

The equation that results is

$$\mathbf{I} = Z_{10} + Z_{20} + Z_{30} + Z_{40},$$

where $\mathbf{I}$ is the identity matrix.

The $f(\lambda) = (\lambda - .181269)$ Equation

The equation that results is

$$\begin{aligned} \mathbf{A} - .181269 &= (.181269 - .181269)Z_{10} \quad (= 0) \\ &+ Z_{11} \\ &+ (.199263 - .181269)Z_{20} \\ &+ (.221199 - .181269)Z_{30} \\ &+ (1 - .181269)Z_{40}, \end{aligned}$$

where $\mathbf{A}$ is the matrix of which the logarithm is desired.

The $f(\lambda) = (\lambda - .181269)^2$ Equation

The equation that results is

$$\begin{aligned} \mathbf{A} - .181269 &= (.181269 - .181269)^2 Z_{10} \quad (= 0) \\ &+ Z_{11} \\ &+ (.199263 - .181269)^2 Z_{20} \\ &+ (.221199 - .181269)^2 Z_{30} \\ &+ (1 - .181269)^2 Z_{40}. \end{aligned}$$

The other matrices of higher powers are developed in the same manner.

Each of the Z matrices contains a maximum of 25 elements. There are five separate Z matrices for a single $\mathbf{A}$ matrix. The number of elements to solve for is thus $5^3 = 125$. The calculation becomes prohibitive very rapidly.

7.4 Summary: It is proposed to use the Discrete Transition Probability Matrix in Place of the Continuous Transition Probability Matrix

It is proposed to use the discrete matrix in place of the continuous matrix because the calculations are simply too prohibitive. The value of this chapter is that the connection between the discrete matrix and the system of continuous differential equations has been demonstrated.

Chapter 8

Summary and Recommendations

8.1 Why The Models Presented Herein Cannot be Used for the Problem Examined for this Project

The Mine group methodology depends upon creating a function, called H or G, that has a single minimum on the range $0 < t \leq \infty$. The functions are reasonably simple to calculate and, since the functions are sums of exponentials, differentiation and subsequent determination of the minimum are also easy. Parenthetically, the aesthetics of their models are most attractive.

Unfortunately for the work proposed in Chapter 7, in the real world one is not guaranteed that the appropriate functions will have the properties ascribed to them that are necessary for the single minimum (of costs) to develop. For instance, the Mine group methodology depends upon the distributions being totally positive of order 2. Total positivity is a non-trivial aspect of distributions. Consider the functions used in the analyses presented in Chapters 5 and 6. For each t the $P_{ij}(t)$ are totally positive of order 2 if

$$\begin{vmatrix} P_{im}(t) & P_{in}(t) \\ P_{jm}(t) & P_{jn}(t) \end{vmatrix} \geq 0 \quad \text{for} \quad \begin{matrix} 0 \leq i < j \leq N \\ 0 \leq m < n \leq N. \end{matrix}$$

For each i the $P_{ij}(t)$ are totally positive of order 2 if

$$\begin{vmatrix} P_{ij}(t) & P_{ij}(u) \\ P_{ik}(t) & P_{ik}(u) \end{vmatrix} \geq 0 \quad \text{for} \quad \begin{array}{l} 0 \leq j < k \leq N \\ 0 \leq t < u \leq \infty. \end{array}$$

For each t the $Q_{ij}(t)$ are totally positive of order 2 if

$$\begin{vmatrix} Q_{im}(t) & Q_{in}(u) \\ Q_{jm}(t) & Q_{in}(u) \end{vmatrix} \geq 0 \quad \text{for} \quad \begin{array}{l} 0 \leq i < j \leq N \\ 0 \leq m < n \leq \infty. \end{array}$$

For each t, $F_i(t)$ is non-decreasing in i. . These four conditions are necessary for there to be a single minimum in the functions H and G. It can be shown that the derivative of the function changes sign only once, and if it changes sign the change is from negative to positive [40].

8.2 What about the Pliska Group?

The Pliska group work is not applicable because of the false positives and negatives. As well, a medical problem has a well defined "end" even if it is an infinite horizon problem (as in having a point mass at infinity). The Pliska group work does not divide into a set of procedures specifying what to do if the system is observed in a given state—the system is either observed to be healthy or deteriorated.

8.3 Recommendations and Future Plans

It is frustrating that, with over 18 months of thinking and working on this project, an appropriate model has not been developed. The models created by the Mine and Pliska groups are wide ranging and have required much effort to program. To

understand the models that were not programmed required nearly as substantial an effort. In this last section of the thesis an outline of what is required to solve a problem on the approximate scale of the NOVA problem is presented.

As an idea of what may not be possible, consider that the Mine group models are constrained in such a manner so as to force the single minimum in the H and G functions. The Mine group models depend on minimizing costs. It was thought to be a relatively simple matter to change the models slightly to maximize rewards instead of minimizing costs. This has, sadly, proved not to be possible because of the characteristics of the models. The benefit of the attempt to model rewards instead of costs was the discovery of what kinds of modifications to the models had to be made in order to use them in a situation such as that presented in Chapter 7.

It should be realized that the general inspection problem that was examined in Chapter 7 can be solved if all that is desired is to determine the next inspection such that the probability of failure of the system is kept below some chosen value. Both the differential equation and matrix multiplication solutions can be implemented—the former for systems with less than about 8 states and the latter for systems with more states. This is similar to what was done in Chapter 3.

In the case where it is desired to prepare an answer as to what to do if the system is observed in a given state, as in the Mine group models, a much more complicated set of model criteria is needed. These criteria are listed in the following subsections.

8.3.1 A State-Dependent Decision

What is desired is a more general state-dependent model in the same class as the Mine group models analyzed for this thesis. A state-dependent decision is the most

important criterion because this creates a truer infinite horizon model than the Pliska group models. An infinite number of repairs must be a possibility. In the Pliska group models only one repair is allowed.

8.3.2 Lack of Total Positivity Criterion

It would be more realistic if the total positivity criterion could be waived. Certainly the operators of a piece of equipment would try as hard as possible to minimize the deterioration rate of their system. Systems in the real world, where there is a large amount of operator interaction will not necessarily decay more rapidly with increasing level of deterioration.

8.3.3 More General Decay Modes

In Chapter 7 a realistic situation of what can be expected from real world measurements of real world systems was presented. The system may decay from one state only to the next highest state, however, as analyzed by the owner of the system, over a discrete time period the system will appear to decay from one state *to all states of greater decay*. Transforming the data into a continuous-time Markov process does not automatically create a simple "one-state-at-a-time" decay chain. The simpler models require a simple decay chain. A model created for a real world system has to be more robust than the Mine group models in that general decay chains are possible.

Another aspect of the Mine group models that can be criticized is the decay mode having the gradual and failure aspects (see Figure 5.1). While some criticism can be directed at that decay chain, it should be noted that the differentiation of

the decay rate into the α and β portions (failure and gradual decay respectively), becomes important in the proofs of the characteristics of the systems they analyze in virtually all of their papers.

8.3.4 More General Cost and Reward Structure

The costs in the Ohnishi, Kawai and Mine algorithm rise with increasing decay. An attempt to replace the costs with a decreasing reward structure, and then placing a minus sign in front of the G formula presented in Chapter 6 in order to maximize rewards, did not prove fruitful. The objective was to create a model based on rewards because the relative magnitude difference between costs and rewards could change the decision to maintain immediately, inspect t years in the future, or to leave the system to fail (i.e. inspect ∞ years in the future).

Including a more general cost and reward structure into the model is an interesting task for future work.

8.3.5 Series Systems

Many of the papers examined during the course of the literature search for this project examined parallel redundant and standby systems. Only one paper, by Kumar, Kapoor and Gupta [22] examined a series system. In this particular paper the series system could be maintained under expensive or inexpensive maintenance. For a large system such as a pipeline the system is obviously a series system with each single corrosion pit, weld section (each about 18 m long), or each kilometre of pipe forming a unique part of the system. A railroad or highway is another example of a series system, as examined by Hatoyama, Fukuoka and Suzuki [9], though in

this particular case the model depends on making decisions after a *regular* inspection is made, i.e. inspections are not planned in the sense of the Mine group optimized inspection time inspections.

Interesting aspects of inspections such as these are the possibility of replacement or maintenance costs depending upon the number of elements replaced at any given time. A pipeline is an obvious candidate for this type of analysis as well.

8.4 Summary of the Method That Must Be Developed

A method must be developed that can analyze optimized inspection times in the same flavour as the Mine group work but that does not depend upon the artificiality of the total positivity of the probability distributions. The desired method must yield a state-dependent decision for machine oriented problems over an infinite horizon—an infinite number of inspections must be possible.

The method should be able to analyze the system in terms of both rewards and costs.

The method should be applicable to a series system such as a pipeline, railroad, highway or other obvious series systems. Parallel system results already exist in the literature in the form of examination of redundant systems.

The method developed should be easily implementable on modern computers in order to make it easy for researchers to analyze real world problems.

Now to solve these new problems...

Bibliography

[1] Anderson, R.F., and A. Friedman, Optimal Inspections in a Stochastic Control Problem with Costly Observations, *Mathematics of Operations Research*, Vol. 2, No. 2, pp. 155-190, 1977.

[2] Anderson, R.F., and A. Friedman, Optimal Inspections in a Stochastic Control Problem with Costly Observations, *Mathematics of Operations Research*, Vol. 3, No. 1, pp. 67-81, 1978.

[3] Barlow, Richard E., and Frank Proschan, *Mathematical Theory of Reliability*, The SIAM Series in Applied Mathematics, John Wiley & Sons, New York, 1965.

[4] Beichelt, F., Minimax Inspection Strategies for Single Unit Systems, *Naval Research Logistics Quarterly*, Vol. 28, No. 3, pp. 375-381, 1981.

[5] Boyd, H., and T.B. Morrison *Answers to Several Questions About Predicting Exterior Corrosion Growth on High Pressure Line Pipe.* Morrison Scientific Problem Solving Report prepared for Department of Design Engineering, NOVA Corporation of Alberta, January 1993.

[6] Boyd, H., G. Haggins and T.B. Morrison, *An Automatic Correlation Methodology for Determining Exterior Corrosion Growth on High Pressure Line Pipe.* Morrison Scientific Problem Solving Report prepared for the Facility Intregity Group of the Field Services Department of NOVA Corporation of Alberta, April 1993.

[7] Chiang, C.L., *An Introduction to Stochastic Processes and Their Applications*, Robert E. Krieger Publishing Company, Huntington, New York, 1980.

[8] Gantmacher, F.R., *The Theory of Matrices*, Volume 1 and Volume 2, Chelsea Publishing Company, New York, N.Y., 1959.

[9] Hatoyama, Y., H. Fukuoka and K. Suzuki, Application of Markovian Decision Theory to the Problem of Highway Maintenance, in *Stochastic Models in Reliability Theory*, Proceedings of a Symposium Held in Nagoya, Japan, April 23-24, 1984, Eds. S. Osaki and Y. Hatoyama. Lecture Notes in Economics and Mathematical Systems, Managing Editors M. Beckmann and W. Krelle, Volume 235, Springer Verlag, New York, pp. 198-212, 1984.

[10] Howard, Ronald A., *Dynamic Programming and Markov Processes*, The M.I.T. Press, Cambridge, Massachusetts, 1960.

[11] Howard, Ronald A., *Dynamic Probabilistic Systems*, Volume 1 and Volume 2, John Wiley & Sons, Inc., New York, 1971.

[12] Kaio, N., and S. Osaki, Some Remarks on Optimum Inspection Policies, *IEEE Transactions on Reliability*, Vol. R-33, No. 4, pp. 277-279, 1984.

[13] Kao, E.P.C., Optimal Replacement Rules when Changes of State are Semi-Markovian, *Operations Research*, Vol. 21, pp. 1231-1249, 1973.

[14] Kander, Z., Inspection Policies for Deteriorating Equipment Characterized by N Quality Levels, *Naval Research Logistics Quarterly*, Vol 25, pp. 243-255, 1978.

[15] Kawai, H., An Optimal Ordering and Replacement Policy of a Markovian Degradation System Under Complete Observation, Part I, *Journal of the Operations Research Society of Japan*, Vol. 26, No. 4, pp. 279-291, 1983.

[16] Kawai, H., An Optimal Ordering and Replacement Policy of a Markovian Degradation System Under Complete Observation, Part II, *Journal of the Operations Research Society of Japan*, Vol. 26, No. 4, pp. 292-307, 1983.

[17] Kawai, H., An Optimal Inspection and Replacement Policy of a Markovian Deterioration System, in *Stochastic Models in Reliability Theory*, Proceedings of a Symposium Held in Nagoya, Japan, April 23-24, 1984, Eds. S. Osaki and Y. Hatoyama. Lecture Notes in Economics and Mathematical Systems, Managing Editors M. Beckmann and W. Krelle, Volume 235, Springer Verlag, New York, pp. 177-186, 1984.

[18] Keller, J. B., Optimum Checking Schedules for Systems Subject to Random Failure, *Management Science*, Vol. 21, No. 3, pp. 256-260, 1974.

[19] Keller, J.B., Optimum Inspection Policies, *Management Science*, Vol. 28, No. 4, pp. 447-450, 1982.

[20] Kirch, R.L.A., and M. Klein, Surveillance Schedules for Medical Examinations, *Management Science*, Vol. 20, No. 10, pp. 1403-1409, 1974.

[21] Kumar, A., and M. Agarwal, A Review of Standby Redundant Systems, *IEEE Transactions on Reliability*, Vol. R-29, No. 4, pp. 290-294, 1980.

[22] Kumar, A., V.B. Kapoor and M.C. Gupta, On Optimal Maintenance of a Series System, *Microelectronics Reliability*, Vol. 23, No. 5, pp. 827-831, 1983.

[23] Lancaster, P. and M. Tismenetsky, *The Theory of Matrices, Second Edition with Applications*, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, Orlando, 1985.

[24] Lincoln, T. L., and G.H. Weiss, A Statistical Evaluation of Recurrent Medical Examinations, *Operations Research*, pp. 187-205, March-April, 1964.

[25] Luss, H., and Z. Kander, Inspection Policies when Duration of Checkings is Non-Negligible, *Operational Research Quarterly*, Vol. 25, pp. 299-309, 1974.

[26] Luss, H., Maintenance Policies When Deterioration Can be Observed by Inspections, *Operations Research*, Vol. 24, No. 2, pp. 359-366, 1976.

[27] Milioni, A. Z., *Optimal Scheduling of Inspections for Systems Under Stochastic Deterioration*, Ph.D. Dissertation, Northwestern University, Evanston, Illinois, June 1987.

[28] Milioni, A. Z., and S. R. Pliska, Optimal Inspection under Semi-Markovian Deterioration: Basic Results, *Naval Research Logistics*, Vol. 35, pp. 373-392, 1988.

[29] Milioni, A.Z., and S. R. Pliska, Optimal Inspection under Semi-Markovian Deterioration: The Catastrophic Case, *Naval Research Logistics*, Vol. 35, pp. 393-411, 1988.

[30] Mine, H., and H. Kawai, An Optimal Maintenance Policy for a 2-Unit Parallel System with Degraded States, *IEEE Transactions on Reliability*, Vol. R-23, No. 2, pp. 81-86, 1974.

[31] Mine, H., and H. Kawai, An Optimal Inspection and Replacement Policy, *IEEE Transactions on Reliability*, Vol. R-24, No. 5, pp. 305-309, 1974.

[32] Mine, H., and H. Kawai, Marginal Checking of a Markovian Degradation Unit when Checking Interval is Probabilistic, *Journal of the Operations Research Society of Japan*, Vol. 19, No. 2, pp. 158-173, 1976.

[33] Mine, H., and H. Kawai, An Optimal Inspection and Maintenance Policy of a Deteriorating System, *Journal of the Operations Research Society of Japan*, Vol. 25, No. 1, pp. 1-14, 1982.

[34] Mokkapati, C., and S. S. Venkata, A Technique for Optimal Sequential Maintenance Scheduling, *IEEE Transactions on Reliability*, Vol. R-30, No. 3, pp. 265-271, 1981.

[35] Morrison, T.B., *Review of Markov Chain Analysis and Statistics of Extremes for use in a Methodology of Analyzing Corrosion Pitting*, prepared for the Department of Design Engineering, NOVA Corporation of Alberta, May, 1989.

[36] Morrison, T.B., *Final Report, RKNOVA, Creation of a Stochastic Model of Corrosion Growth on High Pressure Line Pipe*, Morrison Scientific Problem Solving Report prepared for the Department of Design Engineering, NOVA Corporation of Alberta, June 1991.

[37] Morrison, T.B., and R.G. Worthingham, Reliability of High Pressure Line Pipe under External Corrosion. *Proceedings of the 11th International Conference on Offshore Mechanics and Arctic Engineering*, S.T. Barbas, D. Dall'aglio, M. Fernandez, M. Mohitpour, A.T. Wang and Y.S. Wang, eds., Vol. V, Part B, pp. 401–408, Calgary, Canada, June 7–12, 1992.

[38] Naidu, R. S., and M.N. Gopalan, Analysis of Systems Subject to Inspection and Repair: A State-of-the-Art Survey, *Microelectronics Reliability*, Vol. 24, No. 5, pp. 939-945, 1984.

[39] Ohnishi, M., H. Mine and H. Kawai, An Optimal Inspection and Replacement Policy Under Incomplete State Information: Average Cost Criterion, in *Stochastic Models in Reliability Theory*, Proceedings of a Symposium Held in Nagoya, Japan, April 23-24, 1984, Eds. S. Osaki and Y. Hatoyama. Lecture Notes in Economics and Mathematical Systems, Managing Editors M. Beckmann and W. Krelle, Volume 235, Springer Verlag, New York, pp. 187-197, 1984.

[40] Ohnishi, M., H. Kawai and H. Mine, An Optimal Inspection and Replacement Policy for a Deteriorating System, *Journal of Applied Probability*, Vol. 23, pp. 973-988, 1986.

[41] Osaki, S., and T. Nakagawa, Bibliography for Reliability and Availability of Stochastic Systems, *IEEE Transactions on Reliability*, Vol. R-25, No. 4, pp. 284-286, 1976.

[42] Özekici, S., and T. Papazyan, Inspection Policies and Processes for Deteriorating Systems Subject to Catastrophic Failure, *Naval Research Logistics*, Vol. 35, pp. 481-492, 1988.

[43] Özekici, S., and S. R. Pliska, Optimal Scheduling of Inspections: A Delayed Markov Model with False Positives and Negatives, *Operations Research*, Vol. 39, No. 2, pp. 261-273, March-April 1991.

[44] Paolucci, M., *Computing the Logarithm of a Matrix* Personal Communication to Mr. Morrison, Mike Paolucci, 1992.

[45] Pierskalla, W.P., and J.A. Voelker, A Survey of Maintenance Models: The Control and Surveillance of Deteriorating Systems, *Naval Research Logistics Quarterly*, Vol. 23, No. 3, pp. 353-388, 1976.

[46] Prorok, P.C., The Theory of Periodic Screening I: Lead Time and Proportion Detected, *Advances in Applied Probability*, Vol. 8, pp. 127-143, 1986.

[47] Prorok, P.C., The Theory of Periodic Screening II: Doubly Bounded Recurrence Times and Mean Lead Time and Detection Probability Estimation, *Advances in Applied Probability*, Vol. 8, pp. 460-476-143, 1986.

[48] J.W. Provan and E.S. Rodriguez III, Part I: Development of a Markov Description of Pitting Corrosion, *Corrosion*, Vol. 45, No. 3, pp. 178–192, March, 1989.

[49] E.S. Rodriguez III and J.W. Provan, Part II: Development of a General Failure Control System for Estimating the Reliability of Deteriorating Structures, *Corrosion*, Vol. 45, No. 3, pp. 193–206, March, 1989.

[50] Sengupta, B., Maintenance Policies Under Imperfect Information, *European Journal of Operations Research*, Vol. 5, pp. 198-204, 1980.

[51] Sernik, E.L., and S.I. Marcus, Optimal Cost and Policy for a Markovian Replacement Problem, *Journal of Optimization Theory and Applications*, Vol. 71, No. 1, pp. 105-126, 1991.

[52] Sherif, Y.S., and M.L. Smith, Optimal Maintenance Models for Systems Subject to Failure: A Review, *Naval Research Logistics Quarterly* Vol. 28, pp. 47-74, 1981.

[53] Sherwin, D.J., Inspection Intervals for Condition-Maintained Items Which Fail in and Obvious Manner, *IEEE Transactions on Reliability*, Vol. R-28, No. 1, pp. 85-89, 1979.

[54] Shwartz, M., A Mathematical Model Used to Analyze Breast Cancer Screening Strategies, *Operations Research*, Vol. 26, No. 6, pp. 937-955, 1978.

[55] Sim, S.H., and J. Endrenyi, Optimal Preventative Maintenance with Repair, *IEEE Transactions on Reliability*, Vol. 37, No. 1, pp. 92-96, 1988.

[56] Sondik, E.J., The Optimal Control of Partially Observable Markov Processes over the Infinite Horizon: Discounted Costs, *Operations Research*, Vol. 26, No. 2, pp. 282-304, 1978.

[57] Sorensen, J.D., M.H. Faber, R. Rackwitz and P. Thoft-Christensen, Modelling in Optimal Inspection and Repair, *Proceedings of the 11th International Conference on Offshore Mechanics and Arctic Engineering*, C. Guedes Soares, C.

Ostergaard, M.J. Baker, A. Pittaluga, M. Hunter and P. Thoft-Christensen, eds., Vol. II, pp. 281-288, Calgary, Canada, June 7-12, 1992.

[58] Symbolics, Inc., *MACSYMA Reference Manual*, Computer Aided Mathematics Group, Symbolics Inc., 1988.

[59] Taylor, H. M. and S. Karlin, *An Introduction to Stochastic Modelling*, Academic Press, Inc., Harcourt Brace Jovanovich, Publishers, Orlando, 1984.

[60] Valdez-Flores, C. and R. M. Feldman, A Survey of Preventive Maintenance Models for Stochastically Deteriorating Single-Unit Systems, *Naval Research Logistics*, Vol. 36, pp. 419-446, 1989.

[61] Weiss, G.H., and M. Zelen, A Semi-Markov Model for Clinical Trials, *Journal of Applied Probability*, Vol. 2, pp. 269-285, 1965.

Appendix A

Mine and Kawai MACSYMA Code

```
/* PROGRAM H.MAC, MINE AND KAWAI DYNAMIC PROGRAMMING PROGRAM */
kill(all)$
batchload("c:\\macsyma\\mk\\data.mac")$
batchload("c:\\macsyma\\mk\\f_bar.mac")$
batchload("c:\\macsyma\\mk\\val_det.mac")$
load("c:\\macsyma\\share\\bisect.fas")$
load("c:\\macsyma\\share\\adaplot.fas")$

/* write_tex_file("h.tex"), */

g : 1 $

array(nu, 3) $
nu[2] : 0 $
nu[1] : 0 $
nu[0] : 0 $

/* ALTERNATIVE'S: 1: M, 2: I(inf), 3: I(t) */
alternative_max: 3 $
array(alternative_start, 3) $
alternative_start[2] : 1 $
alternative_start[1] : 1 $
alternative_start[0] : 1 $

/* initialize time */
array(time,3) $
time[2] : 0 $
time[1] : 0 $
time[0] : 7.06 $

cost_repair     : 30  $
time_repair     : 4  $
time_maintain   : 2  $
cost_inspection : 1  $
```

```
cost_maintain   : 20  $ /* 2, 5, 10 and 13 */


test_1 :  alternative_start $
test_2 : -alternative_start $

/* trace(value_determination)$
 * trace(solve)$
 * trace(f_bar)$
 * trace(int_f_bar)$
 * trace(p_ij_nu)$
 */

output : value_determination( g, nu, num_states, test_1, lambda,
         beta, cost_maintain, time_maintain, cost_repair,
         time_repair, cost_inspection, time );

/* g     : part(output,1,1,2);
 * nu[1] : part(output,1,2,2);
 * nu[2] : part(output,1,3,2);
 *
 * g;
 *
 * for i: 1 thru 2 do
 * print('nu[i] = nu[i]);
 *
 */

/* close_tex_file(true); */
/* END OF PROGRAM */
```

```
/* DATA */

array(lambda,3)$
lambda[0] : 10/100 $
lambda[1] : 15/100 $
lambda[2] : 20/100 $

array(beta,3) $
beta[0] : 5/100 $
beta[1] : 5/100 $
beta[2] : 0 $

num_states : 2 $

/* END OF DATA */
```

```
/* VALUE DETERMINATION ROUTINE  */

value_determination( g, nu, n, alternative, lambda, beta,
                     cost_maintain, time_maintain, cost_repair,
                     time_repair, cost_inspection, time ) :=
block( [test_g, v, w, i, j, f_b, f_b_nu, int_f_b, tau, eqn,
        g_out, nu_out, v_temp, w_temp, test_number, nu_temp,
        alternative_chosen, max_alt, solutions_this_time,
        t, h, x, k ],

test_g: g, print("test_g = ", test_g, "g = ", g),

/* initialize the v and w arrays  */
for i:0 thru n do  (
   v[i] : 0,
   w[i] : 0),
for i:0 thru n do
   remvalue ( v[i], w[i] ),

for i:0 thru n do nu[i] : 0,
for i:0 thru n do remvalue(nu[i]),
for i:0 thru n do display(nu[i]),

for i : 0 thru n do (
   if alternative[i] = 1 then (          /* M for maintain */
      v[i] : cost_maintain * time_maintain,
      w[i] : time_maintain
   )
   else if alternative[i] = 2 then (
      /* I(t) = inf for never inspect */
      assume( x > 0 ),
      /* f_b and f_b_nu are zero because the exponentials
       *    are all evaluated at infinity.
       */
      f_b : 0,
      f_b_nu : 0,
      int_f_b : int_f_bar(i, n, lambda, beta, x, 1, time),
      display( f_b, f_b_nu, int_f_b ),
      v[i] :  cost_repair * time_repair * (1 - f_b) +
              cost_inspection * f_b + f_b_nu,
```

```
      w[i] : int_f_b + time_repair * (1 - f_b)
   )
   else if alternative[i] = 3 then (
      /* I(t) for inspect later */
      f_b : f_bar(i, n, lambda, beta, time[i]),

      if i = n then
         answer : read("i = n, this is impossible, type ^C now")
         /* because max_alt at i = n is equal to 2 */
      else if i = n - 1 then
         /* august 16, hard wiring */
         f_b_nu : beta[n-1] * nu[n] * (
                  %e^(-lambda[n-1] * time[i]) /
                  (lambda[n  ] - lambda[n-1]) +
                  %e^(-lambda[n  ] * time[i]) /
                  (lambda[n-1] - lambda[n  ]))
      else /* i <= n - 2 */
         f_b_nu  : p_ij_nu(i, n, lambda, beta, nu, time[i]),

      int_f_b : int_f_bar(i, n, lambda, beta, x, 0, time[i]),

      display( f_b, f_b_nu, int_f_b ),
      v[i] : (cost_repair * time_repair * (1 - f_b) +
            cost_inspection * f_b +
            f_b_nu) / (1 - %e^(-lambda[i] * time[i])),
      w[i] : (int_f_b+time_repair*(1 - f_b)) /
             (1 - %e^(-lambda[i] * time[i]))
   ), /* end of alternative = 3 */
   display(v[i], w[i])
), /* END OF v AND w PART OF EQUATION GENERATION LOOP */

/* remove the values of nu, solve for a new set of nu's */
/* build--up the equations */
for i:0 thru n do remvalue ( g, nu[1], nu[2] ),
for i:0 thru n do (
   if i = 0 then
      nu[0] : 0,
   eqn[i]  : factor(expand(radcan(g * w[i] + nu[i] - v[i]))),
   print(" "),
   display( eqn[i] )
```

```
),

algexact: true, solveradcan: true, globalsolve:true,
solutions_this_time : solve( [eqn[0], eqn[1], eqn[2] ],
                             [ g, nu[1], nu[2] ] ),
g     : part(solutions_this_time,1,1,2),
nu[1] : part(solutions_this_time,1,2,2),
nu[2] : part(solutions_this_time,1,3,2),

print("g = ", g, "test_g = ", test_g,
      "nu[1] = ", nu[1], "nu[2] = ", nu[2]),

/* Evaluation of V_0^A (g^A)
 */

test_v_a : ( cost_repair * time_repair *
             (1 - f_bar(0, 2, lambda, beta, x) )
             + cost_inspection * f_bar(0, 2, lambda, beta, x)
             + p_ij_nu(0, 2, lambda, beta, nu, x) )
             / (1 - %e^(-lambda[0] * x)),

display(test_v_a),

/* if test_g = g then
 *    print(" bingo! The g's match and the optimum
 *          solution has been found! ")
 * else
 *    print(" looking for more optimization "),
 */

/* POLICY IMPROVEMENT PART OF THE ROUTINE */

for i : n step -1 thru 0 do (
   if i = n then max_alt : 2 else max_alt : 3,
   /* max_alt : 3, */
   for j : 1 thru max_alt do (
      if j = 1 then (         /* M for maintain */
         v_temp[j] : cost_maintain * time_maintain,
         w_temp[j] : time_maintain
      )
```

```
      else if j = 2 then (
         /* I(inf) = infinity for never inspect */
         assume( x > 0 ),
         f_b : 0,
         f_b_nu : 0,
         int_f_b : int_f_bar(i, n, lambda, beta, x, 1, time),

         display( f_b, f_b_nu, int_f_b ),
         v_temp[j] : cost_repair * time_repair * (1 - f_b) +
              cost_inspection * f_b + f_b_nu,
         w_temp[j] : int_f_b + time_repair * (1 - f_b )
      )
      else if j = 3 then (
         f_b     : f_bar(i, n, lambda, beta, x),

         /* i will never equal n because max_alt = 2 if i = n */
         if i = n - 1 then
            /* august 16, hard wiring f_b_nu */
            f_b_nu : beta[n-1] * nu[n] * (
                  %e^(-lambda[n-1] * x) /
                  (lambda[n  ] - lambda[n-1]) +
                  %e^(-lambda[n  ] * x)  /
                  (lambda[n-1] - lambda[n  ]))
         else  /* i <= n - 2 */
            f_b_nu  : p_ij_nu(i, n, lambda, beta, nu, x),

         int_f_b : int_f_bar(i,n,lambda, beta, x, 2, time),
         display( f_b, f_b_nu, int_f_b ),

         h : radcan( (  cost_repair * time_repair * (1 - f_b)
                     + cost_inspection * f_b
                     - g * int_f_b
                     - g * time_repair * (1 - f_b)
                     + f_b_nu )
                     / (1 - %e^(-lambda[i] * x)) ),
         display(h),
tex(h),
         xmax : 100,
         xmin : 0,
         ymin : limit(h,x,inf) - 1,
```

```
ymax : limit(h,x,inf) + 1,
adaplot2(h, x, 0.1, 100, ymin, ymax),
kill(xmax, xmin, ymin, ymax),

if i = n then (
   /* limit(h,x,inf) should equal
      v_temp[2] - g * w_temp[2] */
   print("n = ", n, "i = ", i),
   print("limit of h at infinity = ", limit(h,x,inf)),
   print("v_temp[2] - g * w_temp[2] = ",
         v_temp[2] - g * w_temp[2]),
   /* check to see that dH/dt <= 0 for t>= 0 if i = n */
   h_derivative : radcan(diff(h,x)),
   display(h_derivative),
   adaplot2(h_derivative, x, 1, 100, -1, +1)
),

print("limit of h at infinity = ", limit(h,x,inf)),

if i = n then print
   ("you must select option 1 because i = n"),

print("YOUR OPTIONS ARE:
      type 1 to only compare M and I(oo),"),
print("                  H has a minimum: type 2,"),
answer : read("choose a selection"),
print("you chose : ", answer),
if answer = 1 then (
   /* want to compare M and I(oo), so make alt 3 fail */
   v_temp[j] : v_temp[j-1] + 1,
   w_temp[j] : w_temp[j-1] )
else if answer = 2 then (
   h_derivative : radcan(diff(h,x)),
   display(h_derivative),
   adaplot2(h_derivative, x, 1, 100, -1, +1),
   minimum_time : root_by_bisection(h_derivative,x,1,50),
   print("minimum_time = ", minimum_time),
   time[i] : minimum_time,
   h_test : ev(h, x: minimum_time),
   v_temp[j] : h_test,
```

```
            w_temp[j] : 0,
            display(minimum_time, h_test)
         ) /* end of answer = 2, alternative = 3 */
      ),   /* end of alternative 3 */
   display(v_temp[j], w_temp[j])
   ), /* end of j loop */

   for j : 1 thru max_alt do (
      nu_temp[j] : v_temp[j] - g * w_temp[j],
      /* force test_number to equal nu_temp[1] */
      if j = 1 then test_number : nu_temp[1],
      if nu_temp[j] <= test_number then (
         nu[i] : nu_temp[j],
         alternative_chosen : j,
         test_number : nu_temp[j] ),
      print(" max_alt = ", max_alt,
         " i = ", i,
         " j = ", j,
         " nu[",i,"] = ", nu[i],
         " a_chosen = ", alternative_chosen,
         " test_no = ", test_number )
   )     /* bottom of alternative determination loop */
),    /* bottom of policy iteration routine, i loop */

print(" got to bottom of val_det.mac "),
display(g, nu[0], nu[1], nu[2]),

test_v_b : ( cost_repair * time_repair *
             (1 - f_bar(0, 2, lambda, beta, x) )
             + cost_inspection * f_bar(0, 2, lambda, beta, x)
             + p_ij_nu(0, 2, lambda, beta, nu, x) )
             / (1 - %e^(-lambda[0] * x)),

display(test_v_b),
plot(test_v_a, x, 0.01, 100),
plot(test_v_b, x, 0.01, 100),


(g, nu[0], nu[1], nu[2])
)$
```

```
/* END OF VALUE_DETERMINATION
 * ANSWER OF OPTIMUM ALTERNATIVES FOR EACH STATE
 * SHOULD RESULT FROM THIS
 */
```

```
/* f_bar
 */
f_bar(i, n, lambda, beta, time) := (
block[ j, k, index, answer, beta_prod, exp_part, lambda_prod,
       exp_lambda_stuff],
array(answer, 3),
answer[0] : 0,
answer[1] : 0,
answer[2] : 0,
array(exp_lambda_stuff, 3),
exp_lambda_stuff[0] : 0,
exp_lambda_stuff[1] : 0,
exp_lambda_stuff[2] : 0,
for j : i thru n do (
   if j = i then
      answer[j] : %e^(-lambda[i] * time)
   else if j = i + 1 then
      answer[j] : beta[i] * (
                  %e^(-lambda[i] * time) /
                  (lambda[j] - lambda[i]) +
                  %e^(-lambda[j] * time) /
                  (lambda[i] - lambda[j]) )
   else if j > i + 1 then (
      beta_prod : product(beta[k], k, i, j-1),
      /* LOOP OVER I TO J , LEAVING OUT K,
         THE CURRENT ELEMENT OF THE SERIES */
      /* note the cool way the negative signs pop into this */
      for index : i thru j do (
         lambda_prod : 1,
         exp_part : %e^(-lambda[index] * time),
         for k : i thru j do (
            if k # index then
               lambda_prod : lambda_prod *
                             ( lambda[k] - lambda[index] )),
         exp_lambda_stuff[index] : exp_part / lambda_prod
      ),
      answer[j] : beta_prod *
                  sum(exp_lambda_stuff[index], index, i, j)
   ),
   display( answer[j] )
```

```
),
display(sum(answer[j], j, i, n)),
sum(answer[j], j, i, n)
)$
/* END OF f_bar */
```

```
/* p_ij_nu
 *  recall that j starts at i+1
 */

p_ij_nu(i, n, lambda, beta, nu, time) := (
block[k],
array(answer, 3),
answer[0] : 0,
answer[1] : 0,
answer[2] : 0,
/* only way to get here is if i = 0, n = 2 */
print("i = ", i, "n = ", n, "time = ", time),
print("printout of the nu's supplied to p_ij_nu"),
for k : 0 thru 2 do print("k = ", k, "nu[k] = ", nu[k]),

beta[0] * nu[1] *
(%e^(-lambda[0] * time) / (lambda[1] - lambda[0]) +
 %e^(-lambda[1] * time) / (lambda[0] - lambda[1]) )
+
beta[0] * beta[1] * nu[2] *
   (%e^(-lambda[0] * time) /
    ((lambda[1]-lambda[0]) * (lambda[2]-lambda[0]))+
    %e^(-lambda[1] * time) /
    ((lambda[0]-lambda[1]) * (lambda[2]-lambda[1]))+
    %e^(-lambda[2] * time) /
    ((lambda[0]-lambda[2]) * (lambda[1]-lambda[2])))
)$
/* END OF p_ij_nu */
```

```
/* int_f_bar
 * used in the integral of \bar{F}_i(time),
 * to calculate the time of the next expected transition
 */
int_f_bar(i, n, lambda, beta, time_x, int_flag, time) := (
block[ j, k, index, answer, beta_prod, exp_part, lambda_prod,
       temp, t, sex_1, exp_lambda_stuff],
array(answer, 3),
answer[0] : 0,
answer[1] : 0,
answer[2] : 0,
array(exp_lambda_stuff, 3),
exp_lambda_stuff[0] : 0,
exp_lambda_stuff[1] : 0,
exp_lambda_stuff[2] : 0,
for j : i thru n do (
   if j = i then
      answer[j] : %e^(-lambda[i] * time_x)
   else if j = i + 1 then
      answer[j] : beta[i] * (
                  %e^(-lambda[i] * time_x) /
                  (lambda[j] - lambda[i]) +
                  %e^(-lambda[j] * time_x) /
                  (lambda[i] - lambda[j]) )
   else if j > i + 1 then (
      beta_prod : product(beta[k], k, i, j-1),
      /* LOOP OVER I TO J , LEAVING OUT K,
         THE CURRENT ELEMENT OF THE SERIES */
      /* note the cool way the negative signs pop into this */
      for index : i thru j do (
         lambda_prod : 1,
         exp_part : %e^(-lambda[index] * time_x),
         for k : i thru j do (
            if k # index then
               lambda_prod : lambda_prod *
                             ( lambda[k] - lambda[index] )),
         exp_lambda_stuff[index] : exp_part / lambda_prod
      ),
      answer[j] : beta_prod *
                  sum(exp_lambda_stuff[index], index, i, j)
```

```
   ),
   display( answer[j] )
),

temp : sum(answer[j], j, i, n),

assume(x > 0, tau > 0),
if int_flag = 0  then       /* integrate to time[i] */
   integral : integrate(temp, time_x, 0, time[i])
else if int_flag = 1 then   /* integrate to infinity */
   integral : integrate(temp, time_x, 0, inf)
else if int_flag = 2 then (
   /* val of integral as a function of time (x) */
   temp_tau : subst(tau, time_x, temp),
   integral : integrate(temp_tau, tau, 0, time_x)
),

display(integral),

integral
)$
/* END OF int_f_bar */
```

Appendix B

Ohnishi, Kawai and Mine MACSYMA Code

```
/* OKM.MAC, OKM TOTAL AVERAGE COST DYNAMIC PROGRAMMING PROGRAM */

load("c:\\macsyma\\share\\bisect.fas")$
load("c:\\macsyma\\share\\adaplot.fas")$
write_tex_file("h.tex"),

kill(all)$

batchload("c:\\macsyma\\okm\\data.mac")$
batchload("c:\\macsyma\\okm\\int_fbar.mac")$
batchload("c:\\macsyma\\okm\\p_ij.mac")$
batchload("c:\\macsyma\\okm\\m.mac")$
batchload("c:\\macsyma\\okm\\int_a.mac")$

/* begin the main loop of the program here */
for count : 1 thru 20 do (

   print( " top of count loop results, g = ", g),
   for m : 0 thru 3 do (
      print( " m = ", m,
             " alternative[",m,"] = ", alternative[m])),

   for i : n step -1 thru 0 do (            /* state loop */

      x_okm[i] : int_f_bar(i, n, lambda, beta, x,int_flag,time) +
                 p_ij(i, n, lambda, beta, x, x_okm),
      y_okm[i] : m(i, n, lambda, beta, x, inspection_cost) +
                 int_a(i, n, lambda, beta, x, int_flag, time,
                       operating_cost) +
                 p_ij(i, n, lambda, beta, x, y_okm),

      g_okm[i] : (y_okm[i] - g * x_okm[i]) /
                 (1-%e^(-lambda[i] * x)),
```

```
   display(g_okm[i]),

   limit_inf : limit(g_okm[i], x, inf),

   xmax : 100,
   xmin : 0,

/*
   ymin : limit_inf - 2,
   ymax : limit_inf + 2,
   ymin : limit_inf - 0.1,
   ymax : limit_inf + 0.1,

 */
   ymin : limit_inf - 10,
   ymax : limit_inf + 10,

   plot(g_okm[i], x, 1, 100, ymin, ymax),

   print("limit of g_okm[i] at infinity = ", limit_inf),
   print("replacement cost       = ", replacement_cost[i]),
   print("n = ", n, "i = ", i),

   print("YOUR OPTIONS ARE: type 1 to only compare c_i and I(oo),"),
   print("                  type 2 if g_okm[i]  has a minimum,"),
   answer : read("choose a selection"),
   print("you chose : ", answer),

   if answer = 1 then ( /* want to compare i(oo) and c_i */
      if limit_inf < replacement_cost[i] then (
         alternative[i] : 1   /* choose i(oo) */
      )
      else (
         alternative[i] : 2    /* choose c_i  */
      )
   )
   else if answer = 2 then (
      /* want to compare min{g_okm[i]} and c_i */
      g_derivative : radcan(diff(g_okm[i],x)),
```

```
      display(g_derivative),
      xmax : 100,
      xmin : 0,
      ymin : -5,
      ymax :  5,
      plot(g_derivative, x, 1, 100, ymin, ymax),
      minimum_time : root_by_bisection(g_derivative, x, 1, 100),
      print("minimum_time = ", minimum_time),
      time[i] : minimum_time,
      g_test : ev(g_okm[i], x: minimum_time),
      if g_test < replacement_cost[i] then (
         alternative[i] : 3  /* choose h */
      )
      else (
         alternative[i] : 2 /* choose c_i + v_0 */
      )
   )

), /* end of i (state) loop */

print("count = ", count),
print( " completed state loop " ),

/* We need to know alternative[0] because we need to know
 * where to evaluate y_okm[0] and x_okm[0].
 * alternative[0] = 1, then I(oo), so evaluate x and y at oo
 * alternative[0] = 2, then C_i, ratio is oo, so set to 1000
 * alternative[0] = 3, then I(t), so evaluate x and y at t
 */

if alternative[0] = 2 then
   g : replacement_cost[0] /* g should be infinite? */
else if alternative[0] = 1 then
   g : sfloat(ev(y_okm[0], x : inf) / ev(x_okm[0], x: inf))
else if alternative[0] = 3 then
   g : sfloat(ev(y_okm[0], x : minimum_time) / ev(x_okm[0],
       x: minimum_time)),

print("alternative[",i,"] = ", alternative[i], "g = ", g),
```

```
print( " bottom of count loop results "),
for m : 0 thru 3 do (
   print( " m = ", m,
          " alternative[",m,"] = ", alternative[m]))

);/* end of count loop */

/* END OF PROGRAM */
```

```
/* DATA */

n        : 2 $
int_flag : 2$

array(x_okm, 3)$
x_okm[0] : 1$
x_okm[1] : 1$
x_okm[2] : 1$

array(y_okm, 3)$
y_okm[0] : 1$
y_okm[1] : 1$
y_okm[2] : 1$

array(g_okm, 3)$
g_okm[0] : 1$
g_okm[1] : 1$
g_okm[2] : 1$

g : y_okm[0] / x_okm[0] $

array(operating_cost,4) $
operating_cost[0]  : 5  $
operating_cost[1]  : 10 $
operating_cost[2]  : 15 $
operating_cost[3]  : 20 $

array(replacement_cost,4) $
replacement_cost[0]  : 10  $
replacement_cost[1]  : 15  $
replacement_cost[2]  : 20  $
replacement_cost[3]  : 30  $

inspection_cost : 10 $
alpha           : 1/10 $

array(lambda,3)$
lambda[0] : 10/100 $
lambda[1] : 15/100 $
```

```
lambda[2] : 20/100 $

array(beta,3)   $
beta[0] : 5/100 $
beta[1] : 5/100 $
beta[2] : 0     $

num_states : 2 $

array(time,3) $
time[0] : 0   $
time[1] : 0   $
time[2] : 0   $

array(alternative, 4) $
alternative[0] : 0 $
alternative[1] : 0 $
alternative[2] : 0 $
alternative[3] : 0 $

/* END OF DATA */
```

```
/* int_f_bar
 * used in the integral of \bar{F}_i(time),
 * to calculate the time of the next expected transition
 */
int_f_bar(i, n, lambda, beta, time_x, int_flag, time) := (
block[ j, k, index, answer, beta_prod, exp_part, lambda_prod,
       temp, t, sex_1, exp_lambda_stuff],
array(answer, 3),
answer[0] : 0,
answer[1] : 0,
answer[2] : 0,
array(exp_lambda_stuff, 3),
exp_lambda_stuff[0] : 0,
exp_lambda_stuff[1] : 0,
exp_lambda_stuff[2] : 0,
for j : i thru n do (
   if j = i then
      answer[j] : %e^(-lambda[i] * time_x)
   else if j = i + 1 then
      answer[j] : beta[i] * (
                  %e^(-lambda[i] * time_x) /
                  (lambda[j] - lambda[i]) +
                  %e^(-lambda[j] * time_x) /
                  (lambda[i] - lambda[j]) )
   else if j > i + 1 then (
      beta_prod : product(beta[k], k, i, j-1),
      /* LOOP OVER I TO J , LEAVING OUT K,
         THE CURRENT ELEMENT OF THE SERIES */
      /* note the cool way the negative signs pop into this */
      for index : i thru j do (
         lambda_prod : 1,
         exp_part : %e^(-lambda[index] * time_x),
         for k : i thru j do (
            if k # index then
               lambda_prod : lambda_prod *
                             ( lambda[k] - lambda[index] )),
         exp_lambda_stuff[index] : exp_part / lambda_prod
      ),
      answer[j] : beta_prod *
```

```
                    sum(exp_lambda_stuff[index], index, i, j)
   ),
   display( answer[j] )
),

temp : sum(answer[j], j, i, n),
assume(x > 0),
assume(tau > 0),
if int_flag = 0  then       /* integrate to time[i] */
   integral : integrate(temp, time_x, 0, time[i])
else if int_flag = 1 then   /* integrate to infinity */
   integral : integrate(temp, time_x, 0, inf)
else if int_flag = 2 then (
   /* val of integral as a function of time (x) */
   temp_tau : subst(tau, time_x, temp),
   integral : integrate(temp_tau, tau, 0, time_x)
),

display(integral),

integral
)$
/* END OF int_f_bar */
```

```
/* p_ij.mac  */
p_ij(i, n, lambda, beta, time, cost_time) := (
block[ j, k, index, answer, beta_prod, exp_part, lambda_prod,
       exp_lambda_stuff],

/* cost_time is an array that can be y_okm or x_okm */

array(answer, 3),
answer[0] : 0,
answer[1] : 0,
answer[2] : 0,

array(exp_lambda_stuff, 3),
exp_lambda_stuff[0] : 0,
exp_lambda_stuff[1] : 0,
exp_lambda_stuff[2] : 0,
for j : i thru n  do (
   if j = i then
      answer[j] :  cost_time[j] * %e^(-lambda[i] * time)
   else if j = i + 1 then
      answer[j] : beta[i] *  cost_time[j] * (
                  %e^(-lambda[i] * time) /
                  (lambda[j] - lambda[i]) +
                  %e^(-lambda[j] * time) /
                  (lambda[i] - lambda[j]) )
   else if j > i + 1 then (
      beta_prod : product(beta[k], k, i, j-1),
      /* LOOP OVER I TO J , LEAVING OUT K,
         THE CURRENT ELEMENT OF THE SERIES */
      /* note the cool way the negative signs pop into this */
      for index : i thru j do (
         lambda_prod : 1,
         exp_part : %e^(-lambda[index] * time),
         for k : i thru j do (
            if k # index then
               lambda_prod : lambda_prod *
                             ( lambda[k] - lambda[index] )),
         exp_lambda_stuff[index] : exp_part / lambda_prod
      ),
```

```
      answer[j] : beta_prod *  cost_time[j] *
                  sum(exp_lambda_stuff[index], index, i, j)
   ),
   display( answer[j] )
),
sum(answer[j], j, i, n)
)$
/* END OF p_ij */
```

```
/* m.mac
 */
m(i, n, lambda, beta, time, inspection_cost) := (
block[ j, k, index, answer, beta_prod, exp_part, lambda_prod,
       exp_lambda_stuff],
array(answer, 3),
answer[0] : 0,
answer[1] : 0,
answer[2] : 0,
array(exp_lambda_stuff, 3),
exp_lambda_stuff[0] : 0,
exp_lambda_stuff[1] : 0,
exp_lambda_stuff[2] : 0,
for j : i thru n do (
   if j = i then
      answer[j] : %e^(-lambda[i] * time)
   else if j = i + 1 then
      answer[j] : beta[i] * (
                  %e^(-lambda[i] * time) /
                  (lambda[j] - lambda[i]) +
                  %e^(-lambda[j] * time) /
                  (lambda[i] - lambda[j]) )
   else if j > i + 1 then (
      beta_prod : product(beta[k], k, i, j-1),
      /* LOOP OVER I TO J , LEAVING OUT K,
         THE CURRENT ELEMENT OF THE SERIES */
      /* note the cool way the negative signs pop into this */
      for index : i thru j do (
         lambda_prod : 1,
         exp_part : %e^(-lambda[index] * time),
         for k : i thru j do (
            if k # index then
               lambda_prod : lambda_prod *
                             ( lambda[k] - lambda[index] )),
         exp_lambda_stuff[index] : exp_part / lambda_prod
      ),
      answer[j] : beta_prod *
                  sum(exp_lambda_stuff[index], index, i, j)
   ),
```

```
   display( answer[j] )
),
/* display( sum(answer[j], j, i, n) * %e^(-alpha * time) *
 *          inspection_cost),
 * sum(answer[j], j, i, n) * %e^(-alpha * time) *
 * inspection_cost
 */
sum(answer[j], j, i, n) * inspection_cost
)$
/* END OF inspect */
```

```
/* int_a
 * integral of cost of operating beginning in each state i
 * hard-wired for three states (0,1,2)
 */
int_a(i, n, lambda, beta, time_x, int_flag, time,
      operating_cost) := (
block[ j, k, index, answer, beta_prod, exp_part, lambda_prod,
       temp, t, sex_1, exp_lambda_stuff],
array(answer, 3),
answer[0] : 0,
answer[1] : 0,
answer[2] : 0,
array(exp_lambda_stuff, 3),
exp_lambda_stuff[0] : 0,
exp_lambda_stuff[1] : 0,
exp_lambda_stuff[2] : 0,
for j : i thru n do (
   if j = i then
      answer[j] : %e^(-lambda[i] * time_x)
   else if j = i + 1 then
      answer[j] : beta[i] * (
                  %e^(-lambda[i] * time_x) /
                  (lambda[j] - lambda[i]) +
                  %e^(-lambda[j] * time_x) /
                  (lambda[i] - lambda[j]) )
   else if j > i + 1 then (
      beta_prod : product(beta[k], k, i, j-1),
      /* LOOP OVER I TO J , LEAVING OUT K,
         THE CURRENT ELEMENT OF THE SERIES */
      /* note the cool way the negative signs pop into this */
      for index : i thru j do (
         lambda_prod : 1,
         exp_part : %e^(-lambda[index] * time_x),
         for k : i thru j do (
            if k # index then
              lambda_prod : lambda_prod *
                            ( lambda[k] - lambda[index] )),
         exp_lambda_stuff[index] : exp_part / lambda_prod
      ),
      answer[j] : beta_prod *
```

```
                    sum(exp_lambda_stuff[index], index, i, j)
    ),

    assume(x > 0),
    assume(tau > 0),

    /* put the integration here */
    if int_flag = 0  then       /* integrate to time[i] */
       answer[j] : integrate(answer[j], time_x, 0, time[i])
    else if int_flag = 1 then   /* integrate to infinity */
       answer[j]  : integrate(answer[j], time_x, 0, inf)
    else if int_flag = 2 then (
       /* val of integral as a function of time (x) */
       assume(x > 0),
       temp_tau : subst(tau, time_x, answer[j]),
       answer[j] : integrate(temp_tau, tau, 0, time_x)
    ),

    display( answer[j] )
),

sum(answer[j] * operating_cost[j], j, i, n)
)$
/* END OF int_a */
```