PERFORMANCE EVALUATION OF FAULT-TOLERANT SYSTEMS USING TRANSIENT MARKOV MODELS

by

DAVID KEVIN GERBER

S.B., Massachusetts Institute of Technology
(1984)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

MASTER OF SCIENCE IN
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1985

Signature of Author_____
Department of Electrical Engineering and Computer Science
April 19, 1985

Certified by_____
Bruce K. Walker
Faculty Thesis Supervisor

Accepted by_____
Arthur C. Smith
Chairman, Department Graduate Committee

PERFORMANCE EVALUATION OF FAULT-TOLERANT SYSTEMS USING TRANSIENT MARKOV MODELS

by

David Kevin Gerber

Submitted to the Department of Electrical Engineering and Computer Science on April 19, 1985, in partial fulfillment of the requirements for the Degree of Master of Science in Electrical Engineering and Computer Science

ABSTRACT

Fault-tolerant systems and transient Markov models of these systems are briefly explained, and it is suggested that performance probability mass functions (PMFs) are a logical measure of fault-tolerant system performance. For a given Markov model, the operational state history ensemble (OSHE) is the key concept in the first approach to performance evaluation. In certain cases, the growth of the ensemble in time is shown to be linear rather than doubly exponential. A derivation of the v-transform follows. The v-transform is a discrete transform which accurately represents all OSHE behavior and allows symbolic computation of performance PMFs and their statistics. The second approach to performance evaluation uses the theory of Markov processes with rewards. This theory allows more direct computation of certain v-transform results. Comparing the results from the v-transform and from the Markov reward methods provides a check on accuracy and a framework for making reasonable approximations which simplify the computation. Both analytical techniques are applied to systems whose Markov models have from 7 to 50 states, and the effects of model parameter variations on the performance evaluation results will be demonstrated. This thesis concludes with suggestions for future uses of performance evaluation in the areas of fault-tolerant system development and modeling.

Thesis Supervisor: Dr. Bruce K. Walker
Title: Assistant Professor of Aeronautics and Astronautics

ACKNOWLEDGEMENTS

I thank first my thesis supervisor, Bruce Walker, for guiding my work in this new field. Considering his background in this area, I especially appreciate the flexibility he allowed me. This freedom established some connections between previously unrelated fields, and made the work very enjoyable.

The importance of the role that SCHEME played in developing the v-transform concept should not be underestimated. I thank Todd Cass for helping me with programming problems, and I thank Chris Hanson and Bill Rozas of the M.I.T. Artificial Intelligence Laboratory, without whose help and facilities the most interesting results in this thesis would not have been obtained.

I thank the NASA-Langley Research Center for supporting this research through Grant No. NAG1-126 to the M.I.T. Space Systems Laboratory.

Finally, I thank my parents, whose support has always been my greatest asset.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

D	FDI event: Correct detection of a failed component
$\bar{D}$	FDI event: False detection (no failed component present)
$\underline{D}$	FDI event: Missed detection of a failed component
F	General FDI failure event
F_n	Event of one failure among n components
FDI	Failure Detection/Isolation
I	FDI event: Correct isolation of a component
$\bar{I}$	FDI event: Isolation of the wrong component
$\underline{I}$	FDI event: Missed isolation of a component
$\bar{J}(k)$	Total expected performance generated by a transient Markov model in k discrete time steps, starting from state x_1.
$\bar{J}_A(k)$	Expected value of the k-step performance PMF when the Look-Ahead Approximation is used
$\bar{J}_C(k)$	Expected performance of OSHs culled by the Look-Ahead Approximation after k time steps
$\bar{J}_D(k)$	Expected performance defect after k time steps due to the Look-Ahead Approximation
$\bar{J}_j(k)$	Expected value of the k-step performance PMF, given that the system begins in state x_j.
$\bar{J}_{SL}(k)$	Expected k-step performance reaching the system-loss state, given that the system began in x_1.
$\bar{J}_{SLA}(k)$	Expected k-step performance reaching the system-loss state, given that the system began in x_1 and the Look-Ahead Approximation is used
$J(s_k)$	The performance value associated with state s_k (the state occupied on time step k)
$J_{ij}(l,k)$	The cumulative performance of the lth OSH which started in x_j and ended in x_i after k time steps
k	Integral discrete time step argument
k_m	Mission time, expressed as an integral number of FDI test periods

$M(v,k)$	The k-step matrix of v-transforms
$M_j(v,k)$	The column vector which is the jth column of $M(v,k)$
$M_{ij}(v,k)$	The v-transform entry in the ith row and the jth column of $M(v,k)$
MTTF	Mean-Time-To-Failure
n	General discrete time index
OSH	Operational State History
OSHE	OSH Ensemble
P	The single-step transition probability matrix
P_{FA}	FDI false alarm probability
$P(\bullet)$	Probability that the event specified in the parentheses will occur
p_{ij}	The entry in the ith row and the jth column of P
$p_{ij}(l,k)$	The probability that the lth OSH beginning in state x_j and ending in state x_i k time steps later will occur
PMF	Probability Mass Function
r	The vector of rewards or performance values
s_k	The state occupied by a Markov process at time step k
S	The number of states in a Markov model; the rank of the matrix P
SL	System Loss
SLOSH	An OSH which has reached the System-Loss state
$t_{ij}(k)$	The number of distinct OSHs that can possibly begin in state x_j and occupy state x_i exactly k time steps later
$v(k)$	The total performance row vector
$v_j(k)$	The jth entry in the total performance vector
x_j	Designator for model state j
x_1	Designator reserved for the state which represents all redundant components functional and no FDI alarms
x_{SL}	Designator reserved for the system-loss state (also x_S)

ΔT	Discrete time FDI test period
$\phi_{ij}(k)$	The probability of taking any path from state x_j to state x_i in k time steps
$\sigma^2_{J_j}(k)$	Variance of the expected performance of the performance PMF
Σ	The FDI event signifying correct operation of the redundant components and FDI system

CHAPTER 1

INTRODUCTION AND BACKGROUND

1.1 Fault-Tolerant Systems

This thesis focuses on control systems which are highly reliable due to the use of redundant components in their subsystems. The subsystems are redundant in that they have more components (sensors and actuators) than are minimally required to usefully operate the system. The topics of redundant plant components or redundant control computation elements are not considered. The overall system has a design lifetime or mission time during which it operates autonomously and cannot be repaired, and the system is assumed to begin operation with all components fully functional and in use. A redundant control system is necessary because component failures will occur during the life of the plant with non-negligible probability. When components fail, the performance of the system degrades, though it may still be capable of performing its mission. After a critical number or critical combination of components fails, the system degrades to the point where it is impossible for it to continue operating, at which time the system is said to be "lost."

It is intuitively clear that it would be better from a performance standpoint to stop using a failed redundant component rather than to keep using it and allow measurements or control signals to be corrupted. A redundancy management (RM) system is an addition to the control system which is specifically designed to automatically handle such failure contingencies. RM systems generally have two parts. The failure detection

and isolation (FDI) system monitors the components, decides if any have failed, and in the event of a failure detection, decides which component has failed so that it can be isolated from the system. When a failed component has been detected and isolated, the reconfiguration system shuts off the failed component and adjusts the compensation gains to optimize the performance of the closed-loop system using only the remaining components. Figure 1.1 is a block diagram of a plant which includes an RM-based controller.

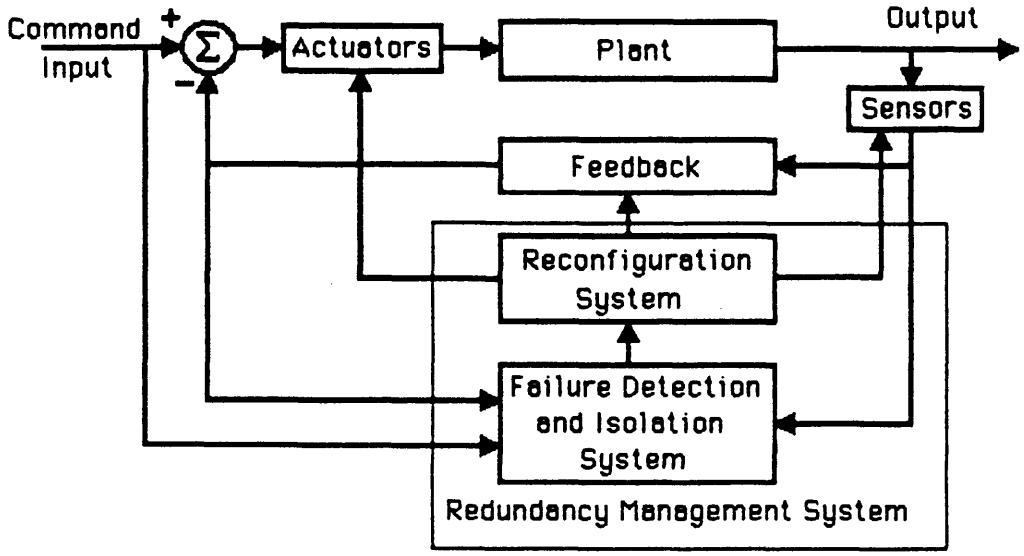


Figure 1.1. Fault-Tolerant System Block Diagram

RM-based control systems are a recent concept. Finding techniques for successfully and efficiently designing RM systems is especially vital to the development of fault-tolerant control systems for complex systems such as large space structures (LSS). RM systems are also widely applicable to flight control systems, inertial navigation systems, and jet engine control systems. Research to date in the LSS area has investigated control component placement considering the likelihood of failures ([3, 11]) and

the development of different FDI strategies ([2, 8, 9]). Current work concentrates specifically on the problems posed by unmodeled plant dynamics in FDI designs. The investigation into reconfiguration techniques is less developed, though general ground rules for control system reconfiguration, as well as strategies for a variety of systems have been suggested [10], and informally discussed.

1.2 Transient Markov Models

Any quantitative discussion of the performance of fault-tolerant systems requires a mathematical model of their behavior. In this respect, previous research [7] has demonstrated Markov models to be superior to other modeling techniques such as combinatorial models. In addition, Markov models have already been used successfully to determine FDI decision thresholds [13] and to predict the accuracy and reliability of redundant systems ([5, 12]). Hence, there exists a good record of experience in using Markov models for various applications. Though the issue of constructing accurate Markov models is not the primary purpose of this thesis, a brief explanation of their construction is included for clarity.

The first step in developing a Markov model is to identify the operational configurations or states of the system. These states depend primarily on the status of the redundant components: How many have failed, which ones, and how many are still in use? A second consideration in identifying states is the FDI system behavior. Though a "good" FDI design will make correct decisions with high reliability, noise, uncertain plant modeling, and other factors make 100% "perfect" decisions impossible. Thus there is the possibility of a misdiagnosis, e.g. detecting a failure when

there really is none, or incorrectly deciding which component has failed. In this way, the decision status of the FDI system will require that more states be included in the model. The number of components and the number of distinct FDI conditions can combine to give the model many states. One simplification that is usually possible is to combine into one "system-loss" state all states which result in a system incapable of performing its mission. Because the system cannot be repaired, it cannot autonomously leave the system-loss state once it enters. Hence, the system-loss state will nearly always be the single trapping state of the model.

A simple example of state identification may be found in Figure 1.2. The states are listed for a generic 4-component system in which 2 components must be working in order for the system to be operational. The simple FDI system cannot indicate a failure if no failure has taken place and isolates detected failures with certainty. Thus the FDI system only adds states in which a failed component is not detected ("uncovered" or "missed"), i.e., states x_2, x_4, and x_5. Note that many states aggregate into the system-loss state. The states are also organized into a Markov state transition diagram under the assumption that only one failure or FDI decision can occur during a single test period.

After identifying all possible operational states of the system, the next step is to determine how the system makes transitions from one state to another. Because the FDI system will be implemented by a computer and have a regular test period, transitions will occur on a discrete time scale. Transition destinations will be probabilistic, because component failures and FDI decisions are governed by stochastic processes. Conditional probabilities that failures, detections, and isolations will

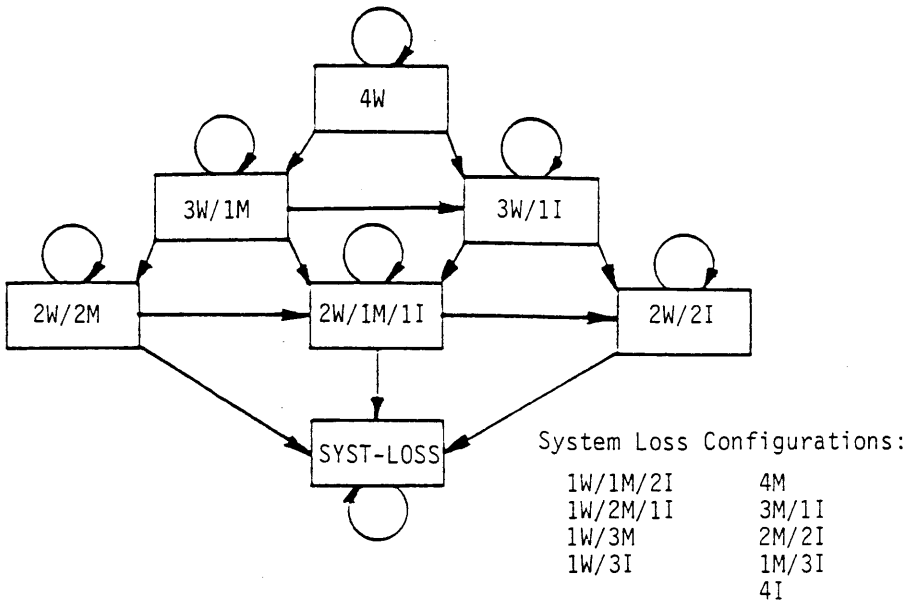


Figure 1.2. Markov Model State Identification

occur in a single test period are well defined because they represent the reliability properties of the components and the design goals of the FDI system. A realistic example of the event tree for a two-stage FDI system appears in Figure 1.3. "D" indicates a detection and "I" indicates an isolation, while overbar indicates that the decision was incorrect and underbar indicates that the decision was missed. In general, all states of the model will also have self-loops.

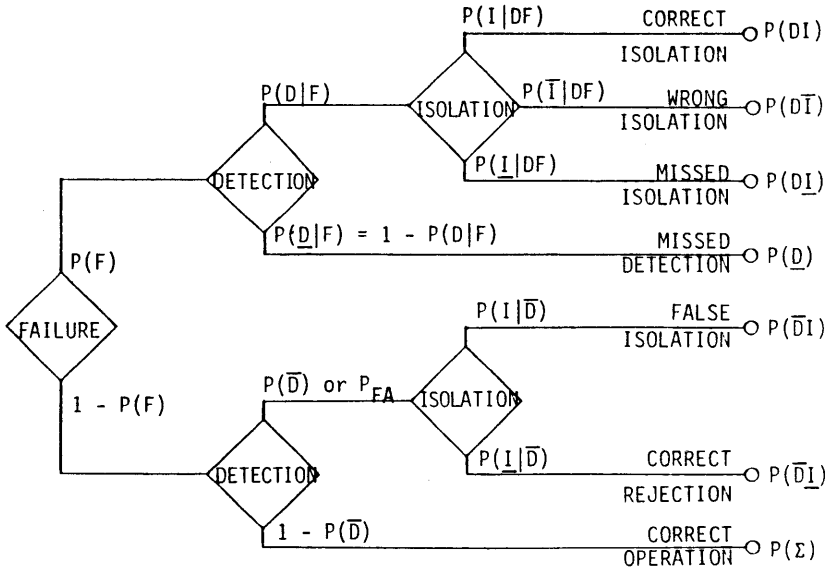


Figure 1.3. Two-Stage FDI Event Tree

In order for the model to be a Markov model, the transitions must be memory-less, that is, the transition probabilities must depend only on the current state of the system. The sequential tests which are sometimes used by FDI systems have memory. However, in this thesis it is assumed that memory can be removed either by adding states to the model or by considering the analysis of a system with memory to be a perturbation of the analysis of a memory-less system.

Markov models of fault-tolerant systems are transient because, after a long enough period of time, all of the components will eventually fail, and the system will be in the system-loss state. Thus, all states except the system-loss state are transient because the probability of occupying those states approaches zero as time goes to infinity. Since the transient states of the model represent the operational states of the modeled system, the performance evaluation problem deals entirely with the transient behavior of these Markov models. There is no interesting steady state behavior.

1.3 Performance Evaluation

Assume that the performance of a simplex plant can be described by a scalar related to the plant's steady-state command following or disturbance rejection capabilities. Such a performance measure is invariant as long as the plant and control law are invariant. In the fault-tolerant case, however, failures of control components, FDI decisions, and reconfiguration actions cause the plant and the control law to change. Fortunately, it can usually be assumed that the closed-loop dynamics quickly stabilize and remain fixed between successive reconfigurations, so that each state of the

Markov model can be assigned a distinct performance value in the manner of a simplex system [4]. These performance values can be calculated for each state a priori given the plant dynamics, redundant component status, and FDI status for that state, and the control law set by the reconfiguration system.

Markov models with state-associated performance values provide a structure which combines the traditional concepts of performance for simplex systems and the stochastic behavior of fault-tolerant systems with redundancy management. It is at this point that the problem of performance evaluation really begins.

1.4 Thesis Goals

The dominant idea in this thesis is that a performance probability mass function (PMF) can logically represent the combination of the state-associated performance measures and the stochastic Markov structure of fault-tolerant systems. Therefore, the performance PMF and the resulting performance statistics should be developed as the standard quantitative measures of fault-tolerant system performance. This concept is also proposed in a paper by Gai and Adams [4]. The first goal of this thesis is to further develop the performance PMF idea by showing how PMFs can be derived for a particular model and by investigating their behavior. Practically speaking, however, such a performance measure is not useful if it cannot be calculated or approximated with a reasonable amount of effort. Therefore, this thesis pursues a second goal of representing performance PMFs in a manner that leads to straightforward computation and solid approximations.

These topics are particularly interesting because of a peculiar sparsity of detailed analyses of transient system behavior in dynamic programming, Markov processes, and operations research oriented literature. Most results and analyses are "steady-state" in one respect or another: steady-state gains, distributions, optimizations etc... This thesis analyzes a transient problem, to the end that parts of the problem which yield steady-state results are selectively defeated so that the important transient behavior is clearly apparent.

This thesis will provide further justification for the use of performance PMFs as a design tool for RM systems. RM systems are already becoming notorious for their myriad design parameters and complexity. Component selection and placement, FDI gains and thresholds, and reconfiguration algorithms are just a few of the engineering challenges inherent to RM system design. Any technique which allows the comparision of two RM systems in terms of risk/benefit tradeoffs or parameter sensitivity will be valuable. Performance PMFs will have the added benefit of demonstrating which approximations are reasonable to make during the modeling of RM-based systems. Performance evaluation results can only be as accurate as the model upon which the analysis is based. Therefore, the ability to obtain meaningful performance information depends upon models that are formulated using reasonable approximations with known effects. Performance PMFs will allow the engineer to gain experience in using different approximations by showing what the effects of the approximations are, thus improving modeling practice.

1.5 Thesis Organization

Chapter Two introduces the concept of operational state history ensembles, discusses their behavior, and shows how these ensembles are the key to developing performance PMFs for a Markov model. The Chapter continues by introducing the v-transform, a new tool for representing and manipulating operational state history ensembles, and shows how to use v-transforms to calculate performance PMFs and their statistics.

Chapter Three embarks on a different course in which some of the v-transform results can be computed more directly through an adaptation of the theory of Markov processes with rewards. The interchange of results between the v-transform and Markov reward methods suggests an approximation with predictable behavior which greatly decreases the computational complexity of the performance PMF problem.

Computational methods and their application to several hypothetical Markov models are presented in Chapter Four to show the typical form of the results of performance evaluation. Chapter Four also demonstrates the effects of varying model parameters such as component reliabilities and false alarm probabilities on the performance evaluation results and shows how to effectively use the approximation described in Chapter Three.

Chapter Five concludes the thesis with a brief summary of the major contributions of this research and recommendations for future work.

CHAPTER 2

PERFORMANCE PMFs AND V-TRANSFORM ANALYSIS

2.1 Operational State Histories

Reference 4 describes operational state histories, but a new explanation will be presented here as background and motivation for the v-transform concept to be presented in Section 2.3. An operational state history (OSH, or trajectory) is a list of the states of the model that a system visits during a specified number of time steps. The listed states must be in the order that the system visits them as determined by the transitions in the model, and each state appears in the list exactly the number of time steps it is visited. Also, unless otherwise specified, the initial state of an OSH is always assumed to be the Markov model state which represents all components functioning with no FDI alarms. If s_k is the state the system occupies at time step n = k, and x_1 is the initial state, then an OSH from n = 1 to n = k would be the list:

$$\{x_1, s_2, s_3, \ldots, s_k\}$$

When the number of time steps is large, it becomes inconvenient (and also unnecessary) to specify an OSH using a long list of states. Fortunately, the important information contained in an OSH can be condensed into two numbers. Each state in the list has a performance value associated with it. Adding these values, or applying any cumulative function to them, provides a cumulative performance value for the OSH. Using a cumulative performance figure is sensible because it reflects the amount of time the system spends in states of varying quality and thus indicates how desirable the OSH is from the performance standpoint.

Second, an OSH does not just specify the states occupied, but also the state transitions, each of which has a probability. The product of these probabilities is the cumulative probability that the system actually follows the path the OSH specifies. After calculating the cumulative probability and performance for an OSH, only the last state occupied needs to be retained because the two cumulative values characterize the entire past history of the process. OSH analysis will be preferable to finding state occupancy statistics, because OSH analysis will provide the PMF of the system performance, a much richer source of information, and can also reflect costs associated with transitions and analyze time varying models.

2.2 OSH Ensembles, PMFs, and Assumptions

An OSH ensemble is the set of all OSHs which a system can possibly follow in a given period of time. Let $t_{ij}(k)$ represent the number of possible OSHs on the interval [0,k] with state at time n=0 $s_0 = x_j$ and the state at time n=k, $s_k = x_i$. For the lth OSH of the set, let $p_{ij}(l,k)$ be the cumulative probability and $J_{ij}(l,k)$ be the cumulative additive performance. If p_{s_{n+1},s_n} is the general single step probability of a transition from state s_n at time n to state s_{n+1} at time n+1, then

$$p_{ij}(l,k) = p_{s_1,s_0} \cdot p_{s_2,s_1} \cdot p_{s_3,s_2} \cdots p_{s_k,s_{k-1}} = \prod_{n=0}^{k-1} p_{s_{n+1},s_n}$$

where $s_0 = x_j$, $s_k = x_i$, and s_1 through s_{k-1} are determined by the lth OSH. Similarly, if $J(s_n)$ is the performance value associated with the state s_n occupied at time n, then

$$J_{ij}(l,k) = J(s_0) + J(s_1) + J(s_2) + \ldots + J(s_k) = \sum_{n=0}^{k} J(s_n)$$

To derive a performance PMF, the first step is to calculate the cumulative probability and performance of every OSH in the ensemble over the interval $[0,k_m]$, where k_m is the mission time of the system. There will be two kinds of OSHs in the ensemble: OSHs which have reached system-loss and OSHs which have ended in one of the other (functional) states. The set of functional OSHs, deconditioned on the final state, specifies the performance PMF of a fault-tolerant system. The OSHs which end in system-loss also provide useful information, but these are not included in the PMF.

Why not simply do these calculations and be finished with the problem? The obstacle lies in the number of OSHs which could typically make up an OSHE. For a rough estimate, assume there are N distinct components in a system which are either "failed" or "not failed" and M binary FDI diagnostics which indicate either "alarm" or "no alarm." The total number of possible Markov states will be on the order of 2^{M+N}, a very large number. Assuming that every state can make a single step transition to any other, then if there are S states, over a period of k time steps, there will be on the order of S^k distinct OSHs! S is large, and k will be also, so exhaustive enumeration and calculation of $p_{ij}(1,k)$ and $J_{ij}(1,k)$ for an entire ensemble is clearly impossible.

Fortunately, a set of very reasonable assumptions ameliorate the problem of OSHE size. As Section 1.2 points out, many Markov model states actually represent a non-functional system and can be aggregated into a single system-loss state. Depending on the model, system-loss aggregation can decrease the number of states by as much as an order of magnitude. Two assumptions reduce the "connectedness" of the remaining states and hence limit OSHE growth. Since the system is autonomous and cannot be repaired,

it is assumed that the system status can only degrade. If the states are ordered appropriately, then the state transition matrix, P, of the Markov model will be lower triangular, or nearly so. (1) Except for self-loops, there will generally be few or no loops in the models. Though the analytical techniques which follow in no way depend upon the structure of the P matrix, it is still good to keep these properties in mind.

The second assumption, also alluded to earlier, is that only one failure or one FDI decision can take place in a single time step. This assumption is reasonable in light of the dichotomy between the typical mean times to failure (MTTF) for components, which are on the order of days to years, and the FDI test period, which is generally 1 second or smaller. This wide difference makes the probability of more than one failure occurring in a single test period and the probability of a failure during a pending FDI decision negligible for the purposes of performance evaluation. This characteristic reduces the connectivity of the model by decreasing the number of states to which the system can transition from a given state in a single step.

Further assumptions which decrease the size of the OSHE originate from the behavior of the OSHs themselves. Before proceeding, however, it will be helpful to have a simple Markov model example which illustrates the behavior about to be discussed. The simple model in Section 1.2 is

---

(1) Systems which can recover from FDI errors will possibly have state transition matrices which are not lower triangular. When the error occurs, the system could transition to a state with much worse performance. When, by some means, the error is later detected and corrected, the system will transition back to a "higher" state with better performance. With appropriate ordering of the model states, every possible occurrence of such behavior would correspond to an entry above the main diagonal of the state transition matrix.

appropriate with the addition of transition probabilities and performance values. These values are somewhat arbitrary, but do reflect a certain structure. For the probabilities, states with no pending FDI decisions (namely, states without M) exhibit high self-loop probabilities, while states which include M have self-loop probabilities which are smaller, but remain larger than the probability of entering the state in the first place. Integral performance values in the lower right hand corner of each box were assigned for reasons to be explained shortly. The higher the value, the worse the performance, and it is always worse to use a failed component than to not use it at all.

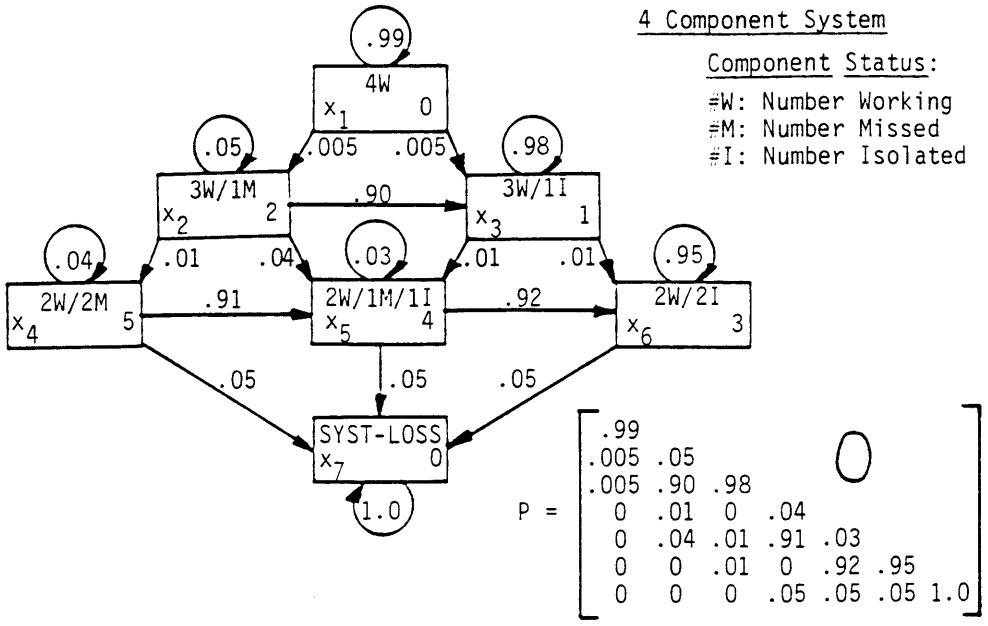


Figure 2.1. Markov Model with Transition Probabilities and Performance Values

Four representative 4-step OSHs for this model will demonstrate all the important characteristics of the OSHE. All of the example OSHs begin in state x_1, and the transition probabilities and performance values for each state, as well as the cumulative probabilities and performances are also shown.

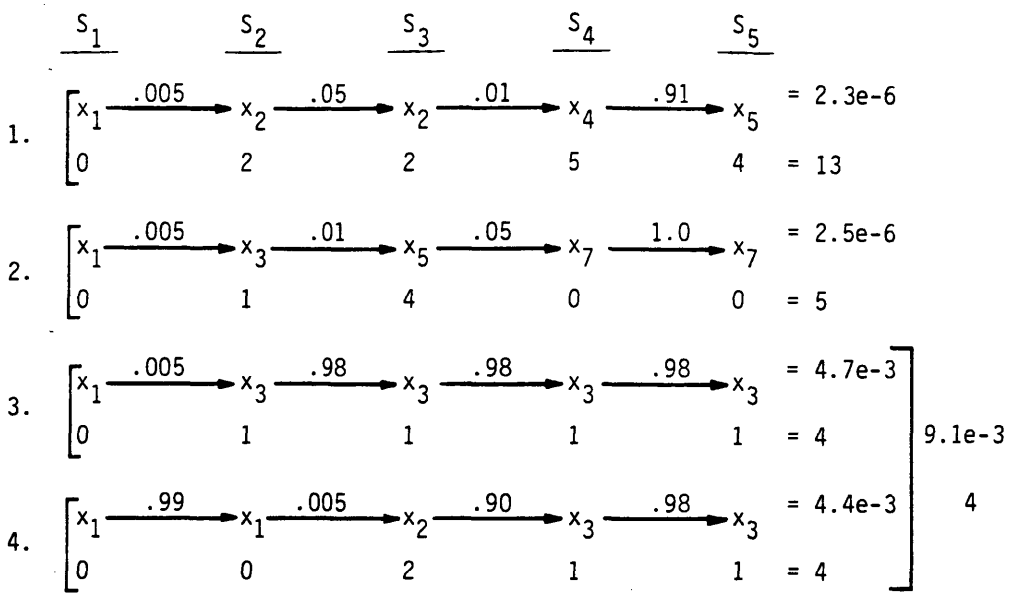


Figure 2.2. Four Example OSHs

The first OSH is a "normal" OSH because it has not reached system-loss during its 4-step lifetime, but has ended in the operational state x_5. The second OSH reaches the system-loss state at time step 3 and thus remains there at step 4. Though the system-loss state has no meaningful performance value, if we assign it the value of zero, then the cumulative performance value of any OSH reaching system-loss will not change thereafter. Since the self-loop probability of system-loss is unity, the cumulative probability of such an OSH will not change either. An OSH is essentially frozen once it reaches the system-loss state, and since we are not as interested in system-loss OSHs (SLOSHs) as in normal OSHs (2) that do not reach system-loss, it makes sense to remove the SLOSHs from the ensemble and treat them separately. Later, there will be other benefits to setting the system-loss performance to zero.

The third and fourth OSHs illustrate the most significant assumption

---

(2) SLOSHs will be useful for reliability evaluation, a by-product of OSH analysis.

of this thesis. Both OSHs begin and end in the same state, have taken different paths to that final state, yet have accumulated the same cumulative performance. As a result, these OSHs can be combined or merged into a single OSH without losing any information, but while decreasing the size of the ensemble. The cumulative probability of the resulting OSH is the sum of the probabilities of the merging OSHs, and the new cumulative performance is the same as that of either OSH.

Merging occurs because the performance values are integers. In general, merging can occur if the performance values of a modeled system are expressed as integral multiples of an arbitrarily small resolution. The resolution of the performance scale can be as small as the accuracy of the analysis requires.

Why is the phenomenon of merging OSHs so important? Earlier, the size of the OSHE was shown to potentially increase exponentially in time (S^k). OSHE growth in the presence of merging is guaranteed to be bounded by a linear function of time. This is a very important result in terms of the computability of performance PMFs. Another look at the example model should provide some insight into why this is true. There is an upper bound on the highest cumulative performance any OSH can have. To find it, identify the OSH which takes the most direct path to the most "expensive" state with a self-loop ($x_1 \rightarrow x_2 \rightarrow x_4$). This OSH has the highest possible cumulative performance, and once it begins to self-loop, the cumulative performance increases linearly at the rate of 5 units per step. All other OSHs must have smaller cumulative performance values and will merge (when deconditioned on the final state) such that there is no more than one OSH for each performance value. Hence, in the worst case, the size of the OSHE will increase linearly in time at the rate of $J(x_*)$ OSHs per time step,

where x_* is the state with the largest performance value.

Now that the concept and utility of OSHEs have been described, it may be apparent that a useful structure for representing and manipulating them is lacking. These considerations are the chief motivation for introducing the v-transform.

2.3 V-Transforms

Using the definitions of $t_{ij}(k)$, $p_{ij}(l,k)$, and $J_{ij}(l,k)$ of Section 2.2, define the v-transform or performance transform, $m_{ij}(v,k)$:

$$m_{ij}(v,k) \equiv \sum_{l=1}^{t_{ij}(k)} p_{ij}(l,k)\, v^{J_{ij}(l,k)}$$

The v-transform is simply a polynomial in v whose terms represent the characteristics of the k-step OSHE between two states. The coefficients are the cumulative probabilities, and the exponents are the cumulative performance values. For example, the transform

$$m_{5,2}(v,3) = 0.20v^2 + 0.50v^4 + 0.20v^7$$

means that the modeled system can transition from state x_2 to state x_5 in 3 steps and have a cumulative performance of 2 with probability 0.2, a cumulative performance of 4 with probability 0.5, and a cumulative performance of 7 with probability 0.2. There remains 0.1 probability that the system transitions to states other than x_5 during those 3 steps.

If k = 1, then for all possible pairs of beginning and end states, $t_{ij}(1) = 1$ if the states connect (non-zero transition probability p_{ij}) or $t_{ij}(1) = 0$ if they don't connect ($p_{ij} = 0$). Thus, define the single-step performance transform, $m_{ij}(v,1)$:

$$m_{i,j}(v,1) \equiv p_{ij} v^{J(x_i)}$$

$J(x_i)$ is the performance value of the destination state, which could be time-varying. The single-step v-transform is always a monomial in v which represents the probability that if the system is currently in state x_j, it will transition in the next step to state x_i with incremental performance $J(x_i)$.

The matrix $M(v,1)$ can be constructed with the single-step transform $m_{ij}(v,1)$ as the entry in the ith row and jth column. $M(v,1)$ is similar to the familiar Markov single-step state transition probability matrix, P. Setting $v = 1$ will yield the coefficient of the single-step v-transform, and therefore:

$$M(v,1)\Big|_{v=1} = P$$

For example, the single-step v-transform matrix for the 7-state model is shown below.

$$M(v,1) = \begin{bmatrix} .99 & & & & & & \\ .005v^2 & .05v^2 & & & & & \\ .005v & .90v & .98v & & & & \\ 0 & .01v^5 & 0 & .04v^5 & & & \\ 0 & .04v^4 & .01v^4 & .91v^4 & .03v^4 & & \\ 0 & 0 & .01v^3 & 0 & .92v^3 & .95v^3 & \\ 0 & 0 & 0 & .05 & .05 & .05 & 1.0 \end{bmatrix}$$

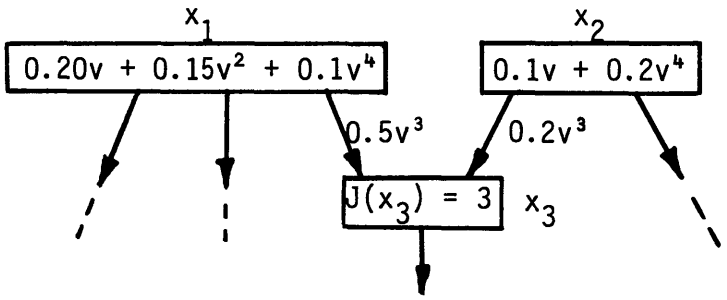
Perhaps the most useful property of P is in finding the k-step transition probability $\phi_{ij}(k)$ associated with transitions from x_j to x_i, or the matrix of these probabilities, $\Phi(k)$:

$$\Phi(k) = P^k$$

When the performance values are time-invariant, $M(v,1)$ has a similar property:

$$M(v,k) = \left[M(v,1)\right]^k$$

The k-step matrix of v-transforms is the kth power of the single-step matrix. As the single-step matrix is raised to higher powers, the monomials in v become larger polynomials. Each term in the polynomial represents one or more OSHs, and the merging property occurs as a natural result of polynomial multiplication: In the resulting polynomial, terms with like exponents are combined. As an example, consider Figure 2.3, where two sets of hypothetical OSHs merge in the state x_3.



$$= 0.1v^4 + 0.075v^5 + 0.05v^7 + 0.02v^4 + 0.04v^7$$

$$= 0.12v^4 + 0.075v^5 + 0.09v^7$$

Figure 2.3. Merging V-Transforms

The matrix of single-step v-transforms provides the framework for representing OSHEs for any specified mission time. Summing the columns of $M(v,k_m)$ yields the performance transforms conditioned only on the initial state x_j:

$$M_j(v,k_m) = \sum_{i=1}^{S-1} \left[M(v,k_m)\right]_{ij}$$

The summation extends only to S-1 because if there are S states and state x_S represents system-loss, the SLOSHs are not included in the performance PMF. If the fully functional initial state of the system is x_1, then

$M_1(v,k_m)$ is the final quantity of interest: The v-transform of the OSHE beginning at state x_1 and propagated over k_m time steps. We are usually interested in $M_1(v,k_m)$, but the performance transforms for all other beginning states can also be found without added difficulty.

Though the performance PMF contains all the system performance information, its statistics can also be helpful performance indicators. The performance transforms has some convenient properties in this regard:

1. k-step transition probability

$$\phi_{ij}(k) = M_{ij}(v,k)\Big|_{v=1}$$

This property is useful because $\phi_{Sj}(k)$ is the unreliability of the system, an informative by-product of the analysis.

2. Expected performance of the k-step OSHE, beginning at state x_j:

$$\overline{J}_j(k) = \frac{dM_j(v,k)}{dv}\Bigg|_{v=1}$$

3. Variance of the expected performance, beginning at state x_j:

$$\sigma^2_{\overline{J}_j(k)} = \frac{d^2M_j(v,k)}{dv^2}\Bigg|_{v=1} - \left[\overline{J}_j(k)\right]^2$$

The system performance is completely characterized by the transform, so arbitrarily high order moments of the performance PMF can be calculated if necessary.

$M_1(v,k_m)$ does not include SLOSHs, but these trajectories still provide useful information. $M_{Sj}(v,k)$ will be the v-transform beginning from state x_j. Once again, we are usually interested in $M_{S1}(v,k_m)$. As above, define the expected performance of the OSHs which have arrived at system-loss (SL denotes system-loss) within k steps:

$$\overline{J}_{SL}(k) \equiv \overline{J}_{S1}(k) = \frac{dM_{S1}(v,k)}{dv}\Bigg|_{v=1}$$

The first column of $M(v,k)$ can be divided into two parts. The sum of the first S - 1 elements of this column is the k-step performance transform starting from state x because the first S - 1 states of the model represent operational states. The derivative with respect to v of this performance transform evaluated at $v = 1$, $\bar{J}_1(k)$, is the expected k-step performance for systems which are still operational after k steps, starting from state x_1. The last element in the first column is called the system-loss transform, and its derivative with respect to v at $v = 1$, $\bar{J}_{SL}(k)$, is the expected performance for those OSHs which reached system-loss during the first k steps starting from state x_1. Recall that the performance of these OSHs is not of interest because the system has not survived for the entire mission. The sum of these two expected performances is the total expected performance, $\bar{J}(k)$, generated by the process in k steps without regard to whether the system survived, starting from state x_1:

$$\bar{J}(k) = \bar{J}_1(k) + \bar{J}_{SL}(k)$$

Chapter Three develops a more direct approach to obtaining $\bar{J}(k)$ using the theory of Markov processes with rewards. Although knowledge of $\bar{J}(k)$ is helpful for checking v-transform results, it alone does not indicate what portion of the total performance is due to SLOSHs and what portion is due to OSHs ending in functional model states. The total expected performance of the system will have other computational uses, however.

Recall that the size of the OSHE, or equivalently, the number of terms in the performance transform, is guaranteed to be proportional to k, and the constant of proportionality is the largest performance value in the

model. If the largest performance value is on the order of 10^3, and the mission length is on the order of 10^5, then there are still 10^8 distinct OSHs to keep track of. This is an improvement over exponential growth, but 10^8 is still large. Except for resolving the performance scale, no accuracy has been lost through approximations. Using the total expected performance values to trade a small loss in accuracy for greatly improved computability will be the goal of Chapter Three.

CHAPTER 3

MARKOV PROCESSES WITH REWARDS

3.1 The Total Performance Vector

This section briefly presents some of the concepts discussed by Howard in [6], but with a strong bias towards the present application.

As explained in Chapter One, two quantities specify the Markov model for evaluating the performance of a fault-tolerant system: the single-step transition probability matrix P and the row vector of state-associated performance values, assuming they are time-invariant. Let the latter be denoted r (rewards) where $r_j = J(x_j)$. Let v(k) be the row vector of total expected performances, indexed according to the starting state, which the Markov process would accumulate if it moved over all possible k-step trajectories. It is possible to find a closed-form expression for v(k) in terms of r, P, and k.

For a single step, v(1) is easy to figure out. v(1) = rP, the sum of the performance values for each state weighted by the probabilities that transitions to those states occur, for each possible initial state. In summation form,

$$v_j(1) = \sum_{i=1}^{S} J(x_i)p_{ij}$$

v(1) is sometimes called the expected immediate reward. To find v(2), note that there is already an expected performance of v(1) accumulated from the first step. To this, add the expected performance accumulated on the second step, namely, r weighted by the probability of being in each state

after 2 steps, given the initial state:

$$v(2) = v(1) + rP^2$$
$$= rP + rP^2$$

Similarly, to find v(3), add to v(2) the expected performance from the third step:

$$v(3) = v(2) + rP^3$$
$$= r(P + P^2 + P^3)$$

By induction, the total expected performance vector given the initial state for an arbitrary number of steps k is easily shown to be:

$$v(k) = r(P + P^2 + P^3 + \ldots + P^k)$$
$$= rP(I + P + P^2 + \ldots + P^{k-1})$$
$$= rP\sum_{n=0}^{k-1} P^n$$

The first entry of v(k) is the total expected performance when beginning in state x_1, the same result obtained at the end of Chapter Two:

$$v_1(k) = \overline{J}(k)$$

3.2 Calculating the Total Expected Performance Vector

The total expected performance vector provides a direct verification of the results obtained through v-transforms, but it can also provide other insight into the behavior of models of fault-tolerant systems. To compute v(k), the modal decomposition of P is helpful:

$$P = V\Lambda W$$

V is the matrix of right eigenvectors of P, Λ is a diagonal matrix of the eigenvalues of P, and W is the matrix of left eigenvectors of P, where $W = V^{-1}$. Substituting the decomposition for P into the expression for v(k):

$$v(k) = rP\sum_{n=0}^{k-1} (V\Lambda W)^n$$

$$= rP\sum_{n=0}^{k-1} V\Lambda^n W$$

$$= rPV\left[\sum_{n=0}^{k-1}\Lambda^n\right]W$$

Since Λ is a diagonal matrix, if λ_i is the ith eigenvalue,

$$v(k) = rPV\ \text{diag}\left[\sum_{n=0}^{k-1}\lambda_i^n\right]W$$

The summation now represents a simple geometric series which can be summed:

$$\sum_{n=0}^{k-1}\lambda_i^n = \frac{\lambda_i^k - 1}{\lambda_i - 1}$$

Thus,

$$v(k) = rPV\ \text{diag}\left[\frac{\lambda_i^k - 1}{\lambda_i - 1}\right]W$$

Using this expression, the total expected performance vector at mission time, $v(k_m)$, can be computed directly.

The total expected performance vector also has interesting asymptotic properties. The eigenvalues of a Markov transition probability matrix have the property that $0 < \lambda_i \leq 1$. Unless $\lambda_i = 1$, then $\lim_{k\to\infty}\lambda_i^k = 0$, and therefore:

$$\lim_{k\to\infty}\left[\frac{\lambda_i^k - 1}{\lambda_i - 1}\right] = \frac{1}{1 - \lambda_i}$$

Therefore, it is possible to find the infinite horizon total expected performance vector, $v(\infty)$:

$$v(\infty) = rPV\ \text{diag}\left[\frac{1}{1 - \lambda_i}\right]W$$

Does $v(\infty)$ have finite or infinite elements? Since the eigenvalue corresponding to the system-loss state is 1, $\frac{1}{1 - \lambda_{SL}}$ is unbounded. But the

performance value for occupation of the system-loss state has been arbitrarily set to zero because its value is of no consequence anyway. The system will be trapped in the system-loss state with probability 1 as k (time) goes to infinity. Setting the system-loss state performance value to zero forces the steady-state gain in cumulative performance for any SLOSH to be zero also. As a result, the steady state gain in the total expected performance represented by the ensemble must also be zero, and all expected performance is accumulated only during transient behavior. Therefore, the elements of $v(\infty)$ converge to finite values which represent the pure transient expected performance values uncorrupted by any steady-state gain.

To avoid computational difficulties in working with the matrix P, it is best to use only the S - 1 by S - 1 upper left partition of P, assuming there are S states and x_S is the system-loss state. This eliminates 1 as an eigenvalue of the resulting reduced order matrix, and then the computer does not have to evaluate (1/0) * 0 when computing $v(\infty)$.

3.3 The Total Expected Performance Profile

The elements of $v(\infty)$ are the total expected performance values which the transient Markov process can generate depending upon the initial state. For those acquainted with dynamic programming, $v(\infty)$ is the same as the vector of relative gains computed during policy iteration, except that in this case they are no longer relative because there is no steady-state gain. The first element of this vector is an upper bound which defines the range of possible expected performance values beginning in state x_1. The expected performance for OSHs arriving at system-loss asymptotically

approaches the infinite horizon total expected performance just as the unreliability of the system approaches unity. Both of these figures as functions of k give an indication of how close the system is to its steady-state. A highly reliable system should be nowhere near its steady-state (which is system-loss) during the duration of a mission. The difference between the total expected performance ($\bar{J}(k)$) and the expected performance for OSHs arriving at the system-loss state ($\bar{J}_{SL}(k)$) is naturally the expected performance contained in the performance PMF, namely $\bar{J}_1(k)$. The relationship between these three quantities can be pictured on an expected performance profile where $\bar{J}_1(k)$, $\bar{J}_{SL}(k)$, and $\bar{J}(k)$ are plotted versus k. Figure 3.1 shows the characteristic shapes of these curves.

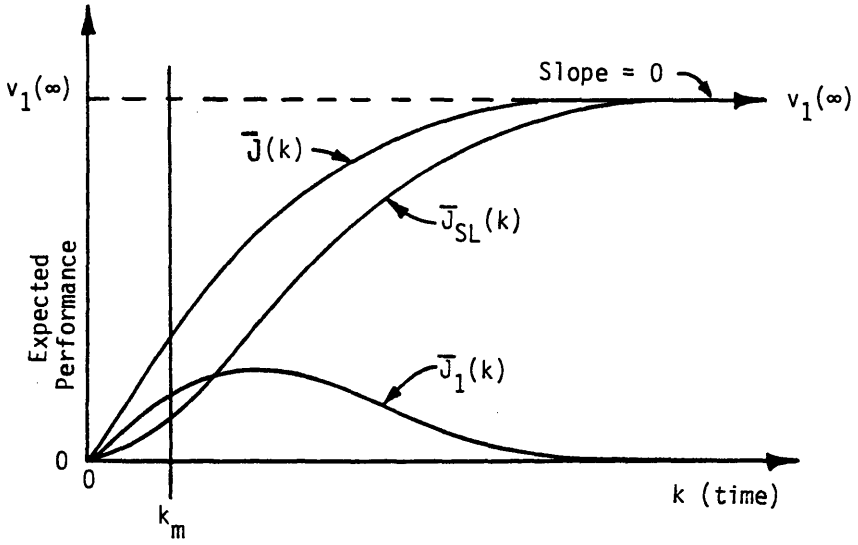


Figure 3.1. Expected Performance Profile

For a reliable system, k_m will lie in the early, sharply rising parts of these curves. The $\bar{J}(k)$ curve can be found using the techniques of this Chapter, but the important breakdown of $\bar{J}(k)$ into its two components $\bar{J}_1(k)$ and $\bar{J}_{SL}(k)$ requires v-transform-generated statistics.

3.4 The Look-Ahead Approximation

Chapter Two ended with a reminder of the complexity of computing performance PMFs using v-transforms, even when the number of terms in the transform of the PMF grows linearly. Combining the total performance vector at mission time $v(k_m)$ with the v-transform structure yields the look-ahead approximation. This approximation was motivated by early v-transform results which indicated that a large number of OSHs represented such a small expected performance that discarding them would affect neither the PMF nor its statistics to a significant degree. The approximation thus reduces the size of the OSHE by eliminating as early as possible during computation those OSHs which will generate an insignificant amount of expected performance. What may be considered "insignificant" must be judged by the engineer, and the chosen value will be shown to have a significant impact on the computability and accuracy of the results.

When manipulating v-transforms, it would seem plausible to examine the expected performance represented by each polynomial term and decide whether to retain the term or discard it based on this value alone. If an OSH has arrived at state x_i from state x_1 after an arbitrary number of steps $\{k: k<k_m\}$ with $p_{i1}(k)$ and $J_{i1}(k)$ as the cumulative probability and performance respectively, then we would examine the quantity:

$$\frac{d}{dv}\left[p_{i1}(k)v^{J_{i1}(k)}\right]\Bigg|_{v=1}$$

Unfortunately, this strategy allows for the possibility of discarding an OSH which could generate a significant amount of expected performance if it remains in the ensemble, even though it has not generated significant expected performance up to the time of the test for its significance.

Using $v(k_m)$, however, it is possible to distinguish between significant and insignificant OSHs (or v-transform terms). $v(k_m)$ provides an upper bound on how much expected performance an OSH could generate during a mission as a function of the starting state. Adding the element of $v(k_m)$ which corresponds to the ending state of the (merged) OSH to the cumulative expected performance of the OSH (i.e. to the exponent of the v-transform term) and then evaluating the derivative at $v = 1$ yields an upper bound on the total amount of expected performance that this term could generate. Thus, knowledge of $v(k_m)$ allows us to "look ahead" and conservatively discard OSHs with the assurance that their future behavior will really be insignificant to the performance results. An OSH can be safely removed from the ensemble if its v-transform term fails the look-ahead test:

$$\frac{d}{dv}\left[p_{ij} \cdot v^{J_{i1}(k)} \cdot v^{v_i(k_m)} \right]\Bigg|_{v=1} \overset{?}{>} \text{tolerance}$$

The tolerance value is set by the system designer, and might typically be some fraction of the total expected performance at mission time starting from state x_1, $v_1(k_m)$. This approximation is conservative because it discards OSHs using the additional expected performance they could generate over an entire mission even though the OSH may already be far into the mission. Therefore many terms are actually retained longer than they need be. In order to discard an OSH earlier, however, requires the computation and storage of the total expected performance vectors for all $\{k: 0 < k < k_m\}$. Then, the remaining expected performance for each state at each step in time would be known exactly. Unfortunately, the extra computation this would require far outweighs the expected benefit.

The expected performance profile changes slightly in light of the look-ahead approximation. Recall that before the approximation is used,

$$v_1(k) \equiv \bar{J}(k) = \bar{J}_1(k) + \bar{J}_{SL}(k) \quad \text{for all } k.$$

With the approximation, there is a certain amount of expected performance, $\bar{J}_C(k)$, represented by OSHs which are culled. For each OSH that is culled, some of the expected performance it represents at the time it is culled would have remained in the PMF and some would have reached the system-loss state had the OSH remained in the ensemble until mission time. Therefore, the expected performance in $\bar{J}_C(k)$ comes directly from $\bar{J}_1(k)$ and $\bar{J}_{SL}(k)$. Let the expected performance in the PMF under the approximation be $\bar{J}_A(k)$ and let $\bar{J}_{SL}(k)$ under the approximation be $\bar{J}_{SLA}(k)$. Then,

$$v_1(k) \equiv \bar{J}(k) = \bar{J}_A(k) + \bar{J}_{SLA}(k) + \bar{J}_C(k) + \bar{J}_D(k) \quad \text{for all } k.$$

The $\bar{J}_D(k)$ is the expected performance defect caused by the approximation. The defect is always present because insignificant OSHs are culled prior to mission time. This means that here is a certain amount of expected performance that would have been generated (whether it ended in system-loss or not) that was not generated due to culling. Under proper use of the approximation, both $\bar{J}_C(k)$ and $\bar{J}_D(k)$ should be very small, and therefore

$$\bar{J}(k) \simeq \bar{J}_A(k) + \bar{J}_{SLA}(k) \quad \text{for all } k.$$

The size of $\bar{J}_D(k)$ can be checked if $\bar{J}_C(k)$ and $\bar{J}_{SLA}(k)$ are calculated during the computation of a PMF. $\bar{J}(k)$ comes from the total expected performance vector, and $\bar{J}_A(k)$ is the expected value of the PMF. Therefore, $\bar{J}_D(k)$ is the only remaining unknown.

The main intent of the look-ahead approximation is that if

$$\bar{J}_A(k) \simeq \bar{J}_1(k)$$

$$\bar{J}_{SLA}(k) \simeq \bar{J}_{SL}(k)$$

then we can be reasonably certain that the resulting performance PMF will be very close to the performance PMF generated without using the approximation. By keeping only the "important" OSHs, the approximation dynamically regulates the OSHE in order to achieve results which have a guaranteed minimum level of significance. This level is determined by the tolerance, and through $\bar{J}(k)$, $\bar{J}_C(k)$, and $\bar{J}_D(k)$, the engineer can find a tolerance that yields accurate results with reasonable computational effort.

This concludes the theoretical portion of this thesis. The past two chapters have often alluded to computational considerations. As the theory was developed, computer programs were simultaneously being written which manipulate v-transform matrices and calculate performance profiles. Chapter Four presents a number of analyzed examples which demonstrate the validity of the theory, show the form of typical results, and reveal some practical considerations in applying these ideas to real problems.

CHAPTER 4

ANALYZED MODELS

4.1 Overview

This chapter presents the results of analyzing several models using the techniques developed in Chapters Two and Three. Section 4.2 discusses two computer programs that have been written to manipulate v-transforms and calculate total expected performance, expected performance profiles, and performance PMFs. The relationship between the characteristics of Markov models and the computational complexity of calculating performance results is also discussed. Section 4.3 presents the several models to be analyzed and discusses why these models were selected and how they were developed. The results from analyzing the models with the computer programs are presented in Section 4.4. Expected performance profiles are presented, and variations in the performance PMF for different numbers of steps k and different tolerances in the look-ahead approximation are examined. Section 4.4 also explores the characteristics of performance evaluation results in two important contexts. First, using performance evaluation results, a 10-state model which approximates a 50-state model is constructed and analyzed, and the results are compared to those of the 50-state model. Second, an 8-state model is analyzed for variations in such parameter values as MTTF and false alarm probabilities to show how such variations in the RM-based fault-tolerant control system are reflected in the performance evaluation results.

4.2 Computation

4.2.1 Symbolic Manipulation of V-Transforms

A listing of procedures for symbolically manipulating matrices of v-transforms is presented in Appendix B in the file APPROX.SCM. These procedures are written in the SCHEME programming language [1], a dialect of LISP, and were compiled and executed on a modified Hewlett Packard 9826U microcomputer. SCHEME (or any LISP dialect) is particularly suited for manipulating v-transforms because of the dynamic nature of the v-transform matrix data structures. During computation, matrix lists can expand and contract dramatically as polynomials are multiplied, merged, and culled using the look-ahead approximation. LISP-related languages are among the few languages in which such structures can be efficiently created and manipulated without the need for direct memory management.

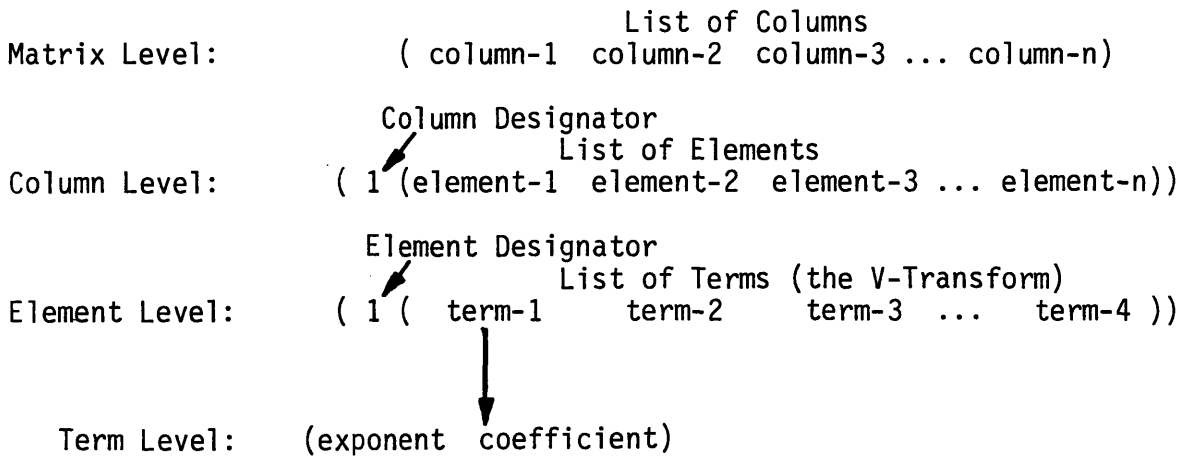


Figure 4.1. Matrix Data Structure

As shown in Figure 4.1, matrices of v-transforms are represented as association lists nested three deep. Association lists are an efficient way to represent tables. Each member in the association list is a pair

with a key (the first item) and an entry (the second item). If the entry is another association list, then multidimensional tables can be represented. On the matrix level, the keys are the column designators and the entries are association lists of elements. On the column level, the keys are the element designators and the entries are association lists of the terms of a v-transform polynomial. Finally, on the element level, the keys are the exponents of the polynomial terms and the entries are the coefficients of the terms.

The matrix manipulation procedures are organized hierarchically according to the level of the matrix structure that each procedure operates on, i.e. matrices, columns, elements, transforms, or terms. The user normally interacts with the top level procedures on the matrix level, though there are some lower-level utilities that are also useful. There are two approaches to generating performance PMFs, and the four top level SCHEME procedures permit these two approaches to be used separately or together. Since all four procedures incorporate the look-ahead approximation, all require the values in the total expected performance vector, $v(k_m)$, and a tolerance value to be specified in their arguments.

The first approach is the linear method. It begins with a column vector which is all zeros except for a single entry of unity in the element position corresponding to the beginning state (usually the first position). The single-step v-transform matrix is then repetitively left-multiplied into this column for a specified number of iterations. The elements of the resulting column are added to give the performance transform. This method is "linear" because the number of matrix-column multiplications required for its execution is equal to the specified mission time (expressed in the number of time samples). Two procedures use the linear approach:

"gen-pmf" and "gen-pmf-with-stats." "Gen-pmf" produces the performance PMF only, while "gen-pmf-with-stats" returns not only the PMF but a sequential listing of a variety of performance statistics ($\bar{J}_1(k)$, $\bar{J}_{SL}(k)$, $\bar{J}(k)$, the unreliability, etc...) for use mainly in performance profiles.

The second approach is the exponentiation method. By multiplying two matrices of v-transforms, this approach directly computes higher powers of the single-step v-transfrom matrix. The "exponentiate" procedure raises any v-transform matrix to a specified power. Thus, to compute the 100th power of the single-step matrix, "exponentiate" can be used to raise the single-step matrix to the 10th power, and then to raise the result to the 10th power again. The performance transform is once again the sum of the elements in the first column of the resulting matrix.

"Exponentiate" and "gen-pmf" each have their advantages and disadvantages. "Gen-pmf" uses less memory space because only the resultant column and the single-step matrix must be stored. "Exponentiate" requires that the entire matrix raised to successively higher powers be stored, and even though we are ultimately interested in only the first column, it is easy to quickly fill up the memory during the analysis of large order models. "Exponentiate" has the advantage of having less overhead for list operations and thus, if the memory is available, high powers can be computed more quickly. "Exponentiate" would thus be the choice for use on computers with virtual memory. "Gen-pmf" has maximum list operation overhead, but if memory is in short supply, "gen-pmf" may be the only option.

The fourth top level procedure combines the two approaches. The procedure "power" takes as one of its arguments a specification list. "Power" uses "exponentiate" to raise the single-step matrix to the powers

listed in the specification list until the last specification is reached. Then "power" uses "gen-pmf" to generate the performance PMF. Thus, the specification list (10 10 72) would compute the performance PMF for a mission time of 7,200 by raising the single-step matrix to the 100th power in two steps and then linearly propagating the first column of the 100-step matrix to the full 7,200 steps. This method has the speed of the "exponentiate" procedure, but saves memory when the exponent is large.

Both approaches to computing performance PMFs are similar with respect to their treatment of SLOSHs and terms culled using the look-ahead approximation (henceforth, "culled OSHs"). Section 2.2 pointed out that the behavior of SLOSHs makes it desirable to remove them from the OSHE. Once SLOSHs are removed from the ensemble, both approaches accumulate the information provided by the SLOSHs as two statistics: the unreliability and $\bar{J}_{SL}(k)$. Culled OSHs are treated similarly, i.e. the total probability and total expected performance represented by these culled OSHs are accumulated and available via the procedure "gen-pmf-with-stats." However, there is a computational expense associated with accumulating the culled and system-loss statistics which can be avoided if the information these statistics supply is not needed. Hence, the procedures in the file FAST.SCM (also in Appendix B) are supplied. These procedures are modifications to selected APPROX.SCM procedures which permit the most direct (and the fastest) computation of performance PMFs.

Once a performance PMF is generated, it is returned in the form of a v-transform, i.e. a list of pairs, each pair consisting of an exponent and a coefficient. At this point, the reliability, expected performance, and variance can be calculated using the procedures "add-coef," "ddvat1," and "variance" respectively. In addition, the PMF can be plotted using the

procedure "plot," and if expected performance statistics have been generated using "gen-pmf-with-stats," they can be plotted using the procedure "profile." See Appendix B for listings and documentation on these procedures.

4.2.2 Computing the Total Expected Performance

A second computer program was written which incorporates the theory of Markov processes with rewards explained in Chapter Three. This straightforward program is written in FORTRAN (Appendix B). The core of the program is the modal decomposition of P (the Markov state transition matrix) using the IMSL routines EIGRF and LINV1F. EIGRF decomposes P into V, the matrix of right eigenvectors, and LAM, the vector of eigenvalues. Then LINV1F computes W, the matrix of left eigenvectors, by inverting V. Using the resulting modal decomposition, the total value vector for mission time and for the infinite horizon is calculated exactly as indicated in the expression for $v(k)$ derived in Chapter Three. A total expected performance profile is also computed and plotted. All other code is primarily for input/output operations. All results can be selectively saved in files for future use.

4.2.3 Computational Complexity

There are several Markov model characteristics which can lead to computational complexity in a performance evaluation problem. Computational complexity most severely affects the time required to generate performance PMFs. First, and most obviously, the more states the Markov model has, the longer it will take to analyze. During v-transform analysis, the increase in computation time comes primarily from the list

operation overhead inherent in SCHEME, whereas in FORTRAN there are simply many more floating point operations to perform. The second key factor is the mission length expressed as an integral number of FDI test periods. A longer mission length will always mean longer computation time for the SCHEME system. However, due to the use of the modal decomposition of P, the FORTRAN system is relatively immune from the effects of long mission times. This is fortunate because the FORTRAN results play a key role in shortening the SCHEME execution time via the look-ahead approximation.

4.3 Models

Several models have been analyzed to demonstrate the different aspects of performance evaluation. They have 7, 8, 10, and 50 states, and they increase in complexity in terms of both the model size and the mission time.

The first model (Figure 4.2) is the model already presented and discussed in Section 2.2. Though it is rather simple, this model comes close to the typical model structure for RM-based fault-tolerant systems. Its dynamics are complicated enough to demonstrate the characteristics of OSHE growth, of the look-ahead approximation, and of performance PMFs.

The second model is a full-scale, realistic model of a tactical system. The control system has two independent component subsystems: the sensors and the actuators. Each subsystem has 4 identical components, any 2 of which must be functional in order for the subsystem to operate. Both subsystems must be operating for the plant to be mission capable.

The FDI systems for each subsystem are identical, independent, two-stage detection/isolation systems with probabilistic behavior similar

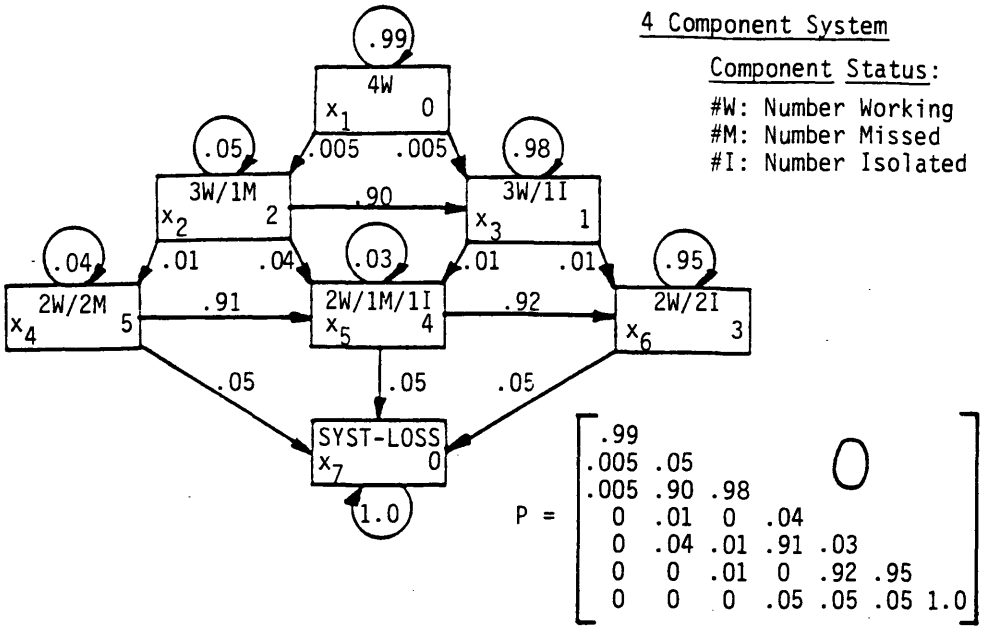


Figure 4.2. 7-State Markov Model

to the event tree in Figure 1.3. The FDI systems combined with the functional component subsets yield an 8-state Markov model for each subsystem which has the "double-star" structure shown in Figure 4.3. The transitions are labelled with the FDI events which cause them to occur. Note that the system can tolerate a single missed detection or a single missed isolation when there are three or four components in use. However, when there are only two components in use, any failure (detected or not) or FDI action will send the system into the system-loss state.

The calculations of all of the transition probabilities are in Appendix A, but a few of the system specifications will be outined here. The system is configured for a tactical mission with a duration of two hours. An FDI test period of 1 second yields a mission length of 7200 discrete steps. Component failure events are assumed to be exponentially distributed with MTTFs of 100 hours for each sensor and 25 hours for each actuator. The sensor and actuator FDI systems are identical, and their

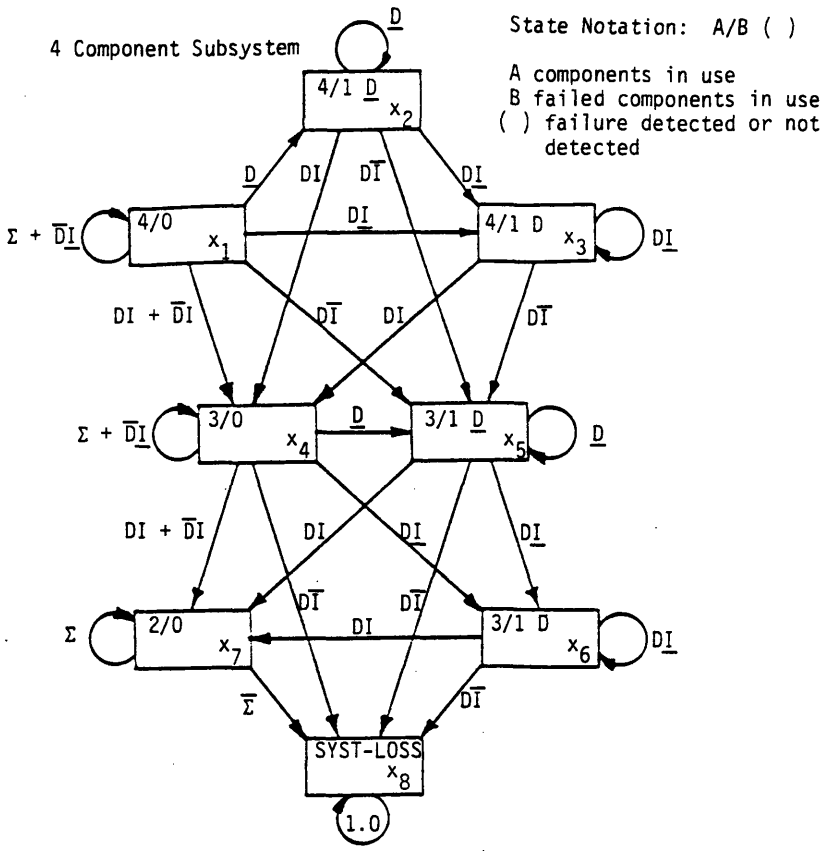


Figure 4.3. Double-Star Markov Model

probabilitistic behavior for each state in the model is summarized in the table below. Entries of N/A are due to two assumptions. If a failure has been detected once, it will certainly be detected again, and a failure cannot occur during a pending FDI decision on a previous failure. The effects of varying the component reliabilities and the false alarm probabilities will be investigated later in Section 4.4.3.

The performance values for an actual system would be based on the closed-loop system performance for each state of the Markov model. In the absence of such values for a real system, a set of performance values were constructed in the following manner: The variance of the noise in a measurement from a sensor was assumed to be σ^2, with a failure causing the standard deviation of the sensor error to increase by 3σ. Measurement

	Event Probability for State:					
	4/0	4/1 D	4/1 $\underline{D}$	3/0	3/1 D	3/1 $\underline{D}$
$P(D\|F)$ =	.99	1.0	.95	.98	1.0	.92
$P(\overline{D})$ =	.001	N/A	N/A	.005	N/A	N/A
$P(I\|DF)$ =	.75	.65	.75	.65	.60	.65
$P(\overline{I}\|DF)$ =	.01	.05	.01	.01	.12	.10
$P(\underline{I}\|DF)$ =	.24	.30	.24	.25	.28	.25
$P(I\|\overline{D})$ =	.005	N/A	N/A	.025	N/A	N/A

Figure 4.4. FDI System Specifications

variances were then computed for each state of the sensor subsystem, normalized by σ^2, and fit to an integral performance scale, yielding sensor subsystem performance values between 1 and 7. The actuator performance was scaled relative to the sensor performance. Actuator subsystem performance values were set an order of magnitude higher than the sensor values because the actuator subsystem is more critical to mission completion than the sensor subsystem is. For actuators, missed detections and missed isolations are particularly dangerous, and therefore a missed detection is assumed to cause the performance to be ten times worse. A missed isolation causes the performance to be only five times worse because it is assumed that the reconfiguration system can aid the performance somewhat just by knowing that a failure is present. The resulting scale of actuator subsystem performance values ranges from 12 to 170. Once again, refer to Appendix A for details.

To complete the full-scale model, the two subsystem models were combined with the assumption that only one FDI event can occur in each subsystem in any single test period. The states of the full-scale model

are the set of composite states formed by selecting all possible pairs of states, taking one state from each subsystem model. Since each subsystem has 8 states, there would be 64 such pairs, but this drops to 50 pairs when all pairs containing a system-loss state are aggregated.

The task of calculating the transition probabilities for the resulting 50 states is too difficult to do by hand, so it has been automated in the computer programs. The front end of the FORTRAN code computes the resulting single-step transition matrix P given two input subsystem matrices P1 and P2. In the SCHEME system, the procedures in the file EXPANDER.SCM perform this task (Appendix B). The top-level procedure is "make-matrix," which takes two arguments, matrix1 and matrix2, the single step v-transform matrices for the two subsystems. During the process of combining the subsystems, the performance values for the resulting composite states are the sums of the performance values of the two component subsystem states.

The 50-state model is large, complex, and representative of the kind of models which could easily be encountered during the performance evaluation of real systems. The dynamics are much slower, and the mission time is much longer than in the two preceding models. This model is analyzed to investigate the limits of the computational techniques implemented in the FORTRAN and SCHEME computer programs. The analysis also suggests ways to overcome these limits by simplifying the model, as Section 4.4.2 will demonstrate.

4.4 Performance Evaluation Results

4.4.1 The 7-State Model

Using the FORTRAN program, the infinite horizon total performance vector, $v(\infty)$, was computed:

$$v(\infty) = \begin{bmatrix} 110.27 & 108.04 & 109.52 & 58.06 & 57.00 & 56.97 & 0 \end{bmatrix}$$

A short listing of the FORTRAN results for this model is presented in Appendix C. The first element of this vector, $v_1(\infty)$, represents the total expected performance that the system can generate in an unlimited amount of time, given that it began in state x_1. The last element corresponds to the expected performance that would be generated if the system began in the system-loss state and stayed there. This entry will always be zero since the performance value of the system-loss state is chosen as $J(x_{SL}) = 0$.

An arbitrary mission time of $k_m = 150$ was selected for this system. At mission time, the total expected performance vector is

$$v(k_m) = \begin{bmatrix} 62.73 & 101.16 & 102.41 & 58.04 & 57.00 & 56.97 & 0 \end{bmatrix}$$

Note that $v_1(k_m) < v_1(\infty)$. When calculating performance PMFs, $v_1(k_m)$ is a guideline to defining an appropriate tolerance for the look-ahead approximation because it is the upper bound on the total expected performance the system can generate during a mission time. Using $v_1(k_m)$ as this limit rather than $v_1(\infty)$ is preferable because in most systems $v_1(k_m) << v_1(\infty)$. Figure 4.5 is the total expected performance profile for $v_1(k)$ plotted from $k = 0$ to $k = 1000$. This model effectively reaches steady-state in approximately 750 time steps because at this time, its unreliability is very close to 1, and the total expected performance $v_1(k)$ is very close to $v_1(\infty)$. The plot of $v_1(k)$ approaches the asymptote $E(J) = 110.27$ showing that the steady-state gain in expected performance is

zero because $J(x_{SL})$ is zero. All expected performance is generated during occupancy of the transient states of the model. If $J(x_{SL})$ had been a non-zero constant, the steady-state performance profile would have been a ramp with a slope equal to that constant, and the transient behavior would thus be obscured.

Using the SCHEME system, several performance statistics were calculated in order to check the accuracy of the SCHEME results against those of the FORTRAN system. These values were generated using a tolerance of 10^{-10} for the look-ahead approximation.

		SCHEME	FORTRAN
$v_1(k_m) = \bar{J}(150)$:		62.7362	62.7363
$v_1(\infty) \simeq \bar{J}(750)$:		110.13	110.14
Unreliabilities	$\phi_{S1}(150)$:	0.47009	0.47009
	$\phi_{S1}(750)$:	0.998427	0.998463

The expected performance represented by the OSHs culled by the look-ahead approximation, $\bar{J}_C(k)$, is 6.3×10^{-2} after 750 time steps. The expected performance and cumulative probability represented by the culled OSHs will generally account for any small discrepancies between the SCHEME and FORTRAN results, if the look-ahead approximation is used properly. In this case, the tolerance is small enough that the effects of the approximation on the results are negligible.

The SCHEME system divides the total expected performance $\bar{J}(k)$ or $v_1(k)$ into its two major components, $\bar{J}_1(k)$ and $\bar{J}_{SL}(k)$. Figure 4.6 is an expected performance profile and Figure 4.7 is the system unreliability curve on the same time scale, both generated by the SCHEME system. In Figure 4.6, the top curve, $\bar{J}(k)$, is the total expected performance profile, as computed by

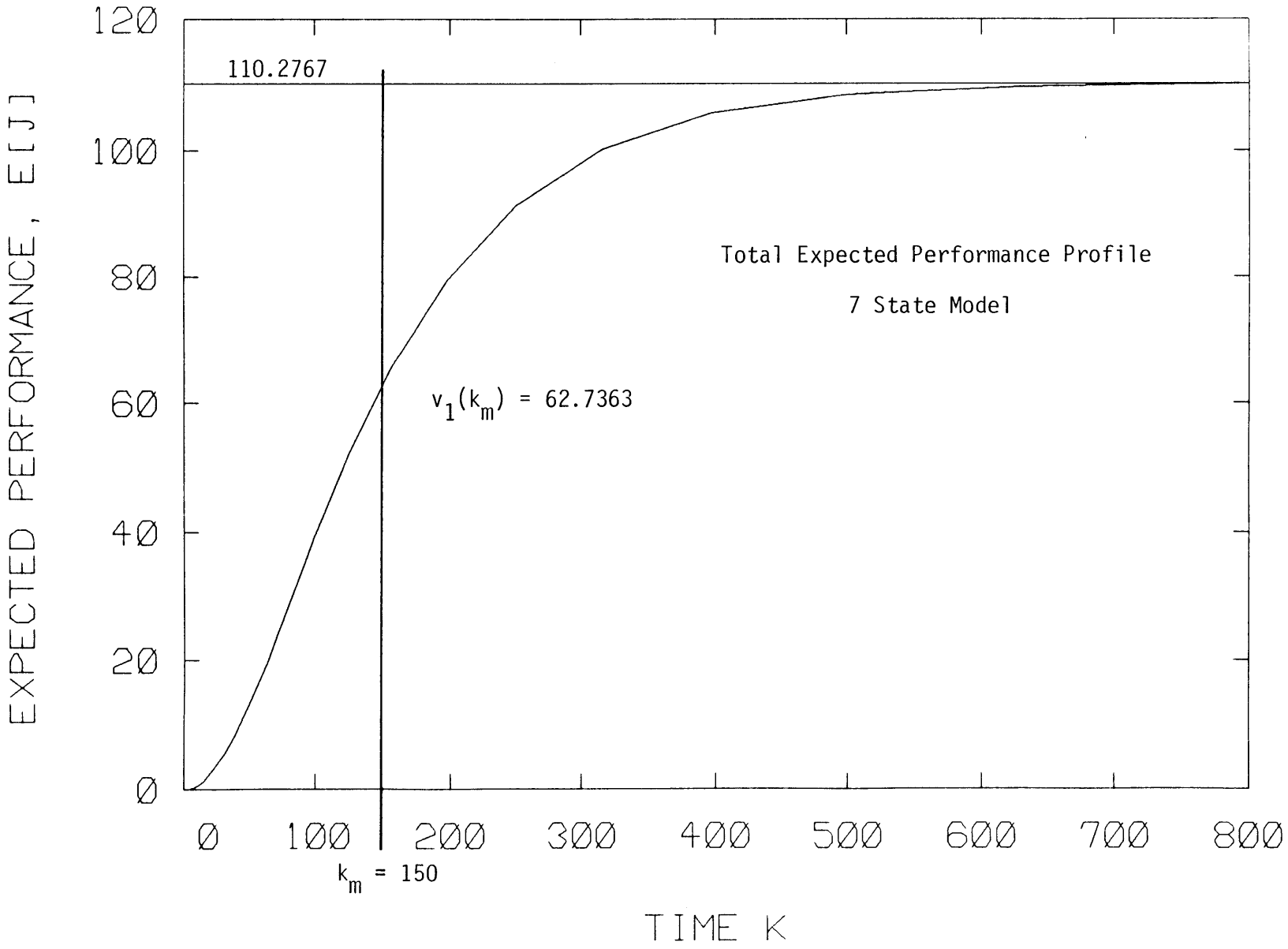


Figure 4.5. Total Expected Performance Profile for the 7-State Model

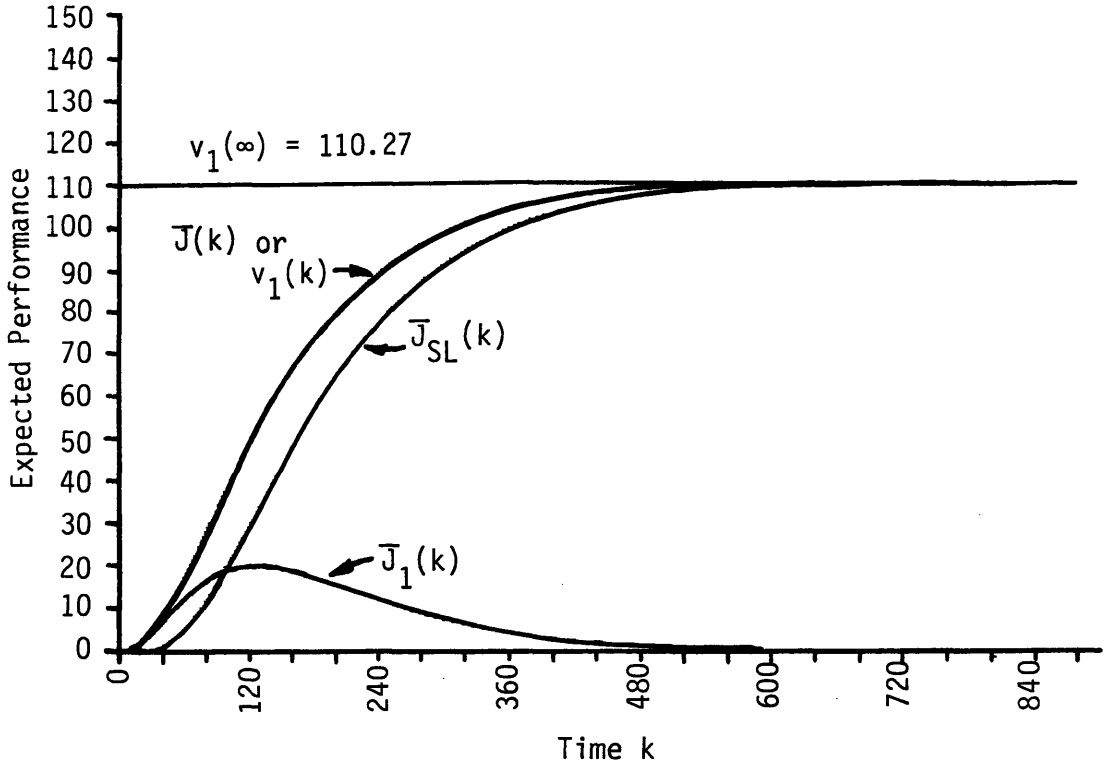


Figure 4.6. Expected Performance Profiles for the 7-State Model

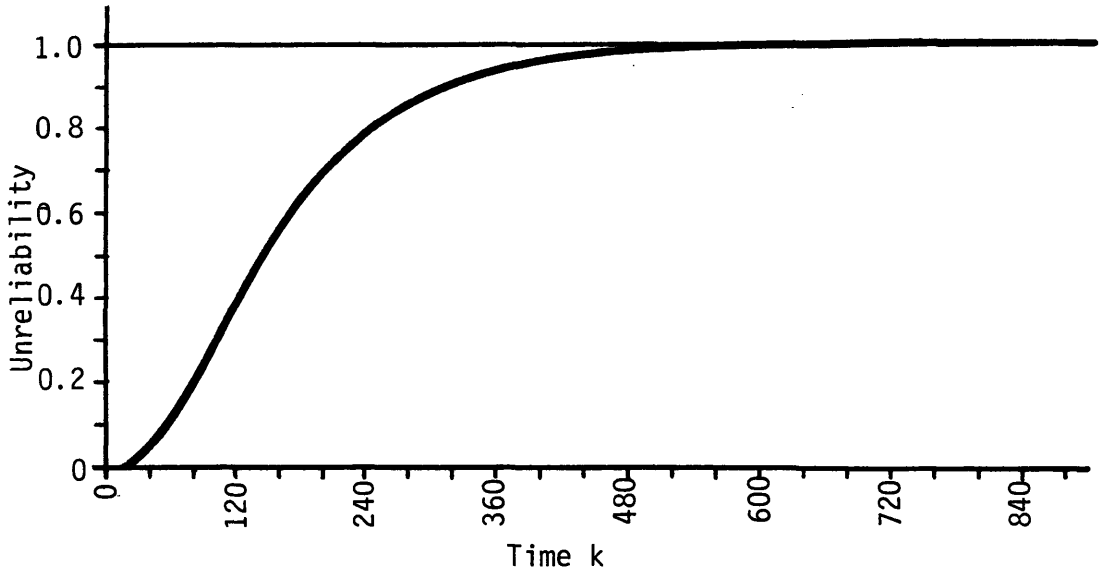


Figure 4.7. Unreliability Profile for the 7-State Model

the FORTRAN system in Figure 4.5. The other two curves, $\bar{J}_{SL}(k)$ and $\bar{J}_1(k)$, are the components which add to produce $\bar{J}(k)$. $\bar{J}_{SL}(k)$, the expected performance represented by SLOSHs, asymptotically approaches $\bar{J}(k)$, and $\bar{J}_1(k)$, the expected value of the performance PMF, asymptotically approaches zero as k increases. This behavior was predicted in Chapter Three. The similarity in the shapes of the unreliability and $\bar{J}_{SL}(k)$ curves shows that all of the probability and expected performance "flows" with the OSHs toward the system-loss state as k becomes large.

Before presenting performance PMFs, the structure of the PMF plots needs some explanation. All performance PMFs will be plotted in the same format. The horizontal axis is the performance axis and the vertical axis is the logarithmic probability axis. PMFs are plotted as a series of impulses whose horizontal location is determined by the cumulative OSH performance and whose vertical height is determined by the cumulative OSH probability. The PMFs appear solid due to the graphics resolution limits. The horizontal axis has a range of 450 pixels, and the impulses are plotted as densely as possible in order to present more information. Because the range of the PMF performance scale generally exceeds 450, 2 or more neighboring impulses are added to create the impulse which is actually plotted. Adding impulses has little effect on the shape of the PMF because the probability axis is logarithmic, and therefore the plot accurately represents the envelope of the PMF that would result if higher resolution graphics had been used. The number of impulses combined into one impulse affects the horizontal axis scaling and will be noted on each PMF. Where comparisons are made, this scaling will be identical for all PMFs under consideration.

The vertical axis is the logarithm of the cumulative OSH probability.

A logarithmic scale was chosen in order to portray the wide range of probabilities typically encountered in the PMFs. The scale on all plots is from 10^{0} to 10^{-15}. It is assumed that events which occur with a probability of less than 10^{-15} are not of interest and will be culled by the look-ahead approximation.

It is important to note that although these plots are performance PMFs, the probability they represent does not add up to 1.0. The total probability in the performance PMF represents the system reliability, and the difference between this probability and 1.0 is represented mostly by the SLOSHs that were removed from the ensemble. As a result, no PMF is complete without an accompanying specification of either the reliability or unreliability of the system. It would be possible to normalize the PMF by the system reliability, but this impedes direct comparison of two or more systems that have different reliabilities.

Performance PMFs for the 7-state model were calculated to demonstrate two phenomena: The variation in the PMF shape for increasing times k and for different look-ahead tolerances. These examples will provide a basis for examining the PMFs of more complicated systems.

Figures 4.8 - 4.12 are the performance PMFs for the 7-state model at k = 50, 100, 150, 250, and 750 respectively. The look-ahead approximation used the v(150) vector computed by the FORTRAN program and a tolerance of 10^{-6}. That this tolerance has a negligible effect on the shape of the performance PMF will be demonstrated shortly. All of these plots have the same performance scale, and 2 impulses of the PMFs were combined to generate each impulse in the plots (Scale = 2). These PMFs were produced by the procedure "gen-pmf-with-stats" so that the $\bar{J}_{SL}(k)$ statistic could be shown with each plot.

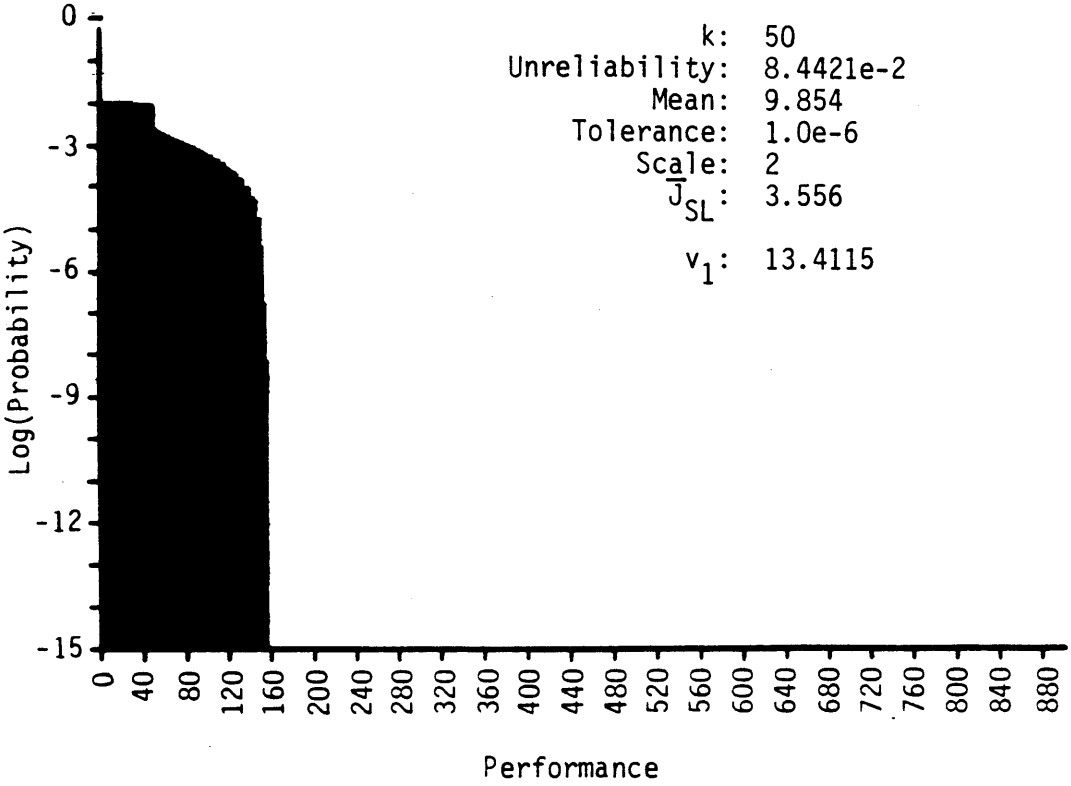


Figure 4.8. 50-Step Performance PMF for the 7-State Model

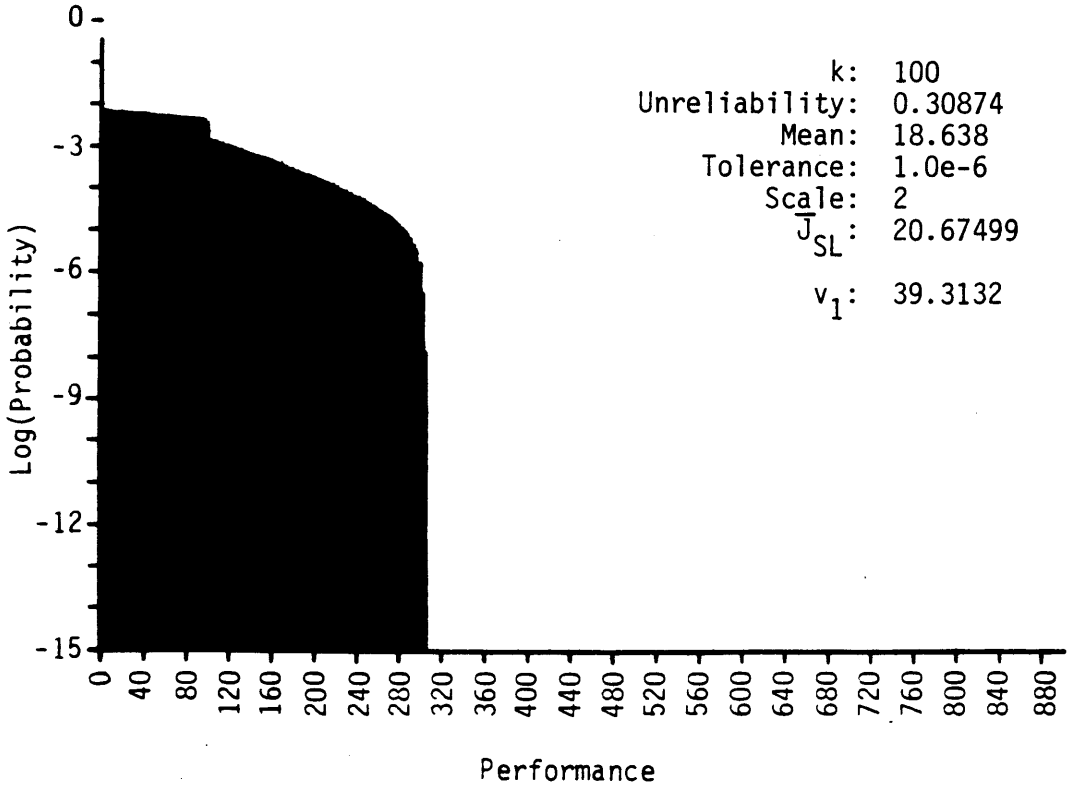


Figure 4.9. 100-Step Performance PMF for the 7-State Model

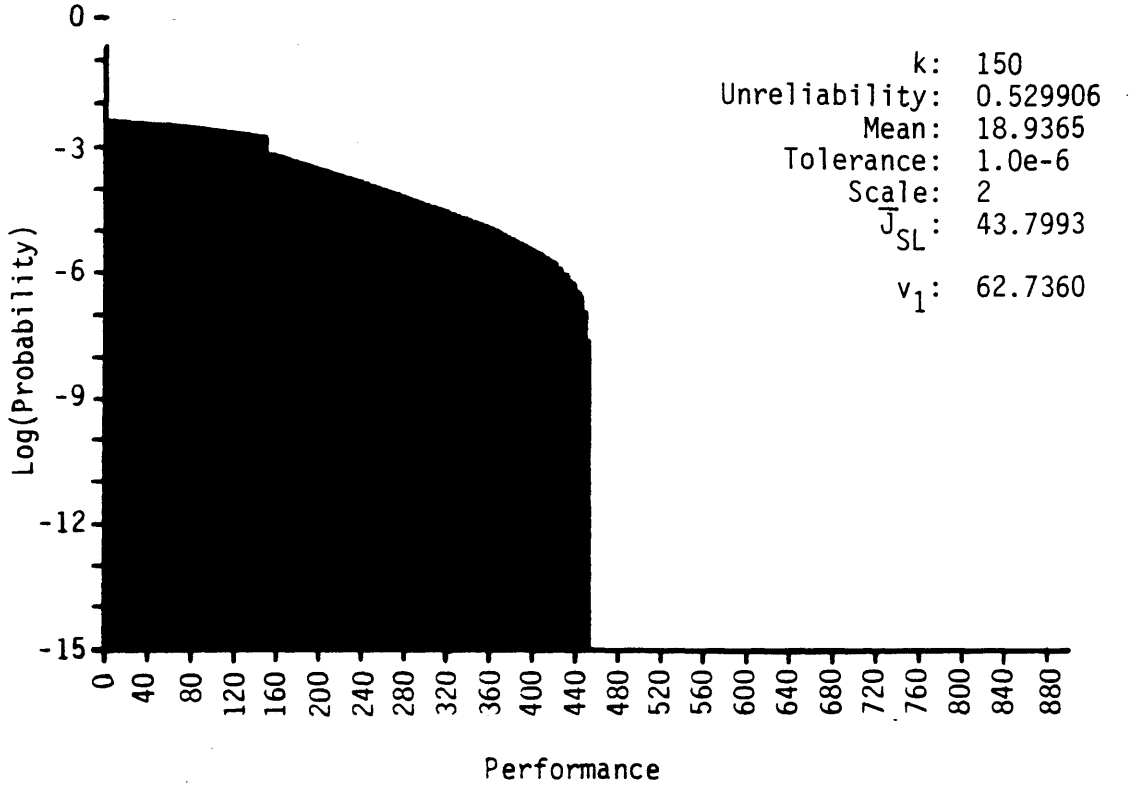


Figure 4.10. 150-Step Performance PMF for the 7-State Model

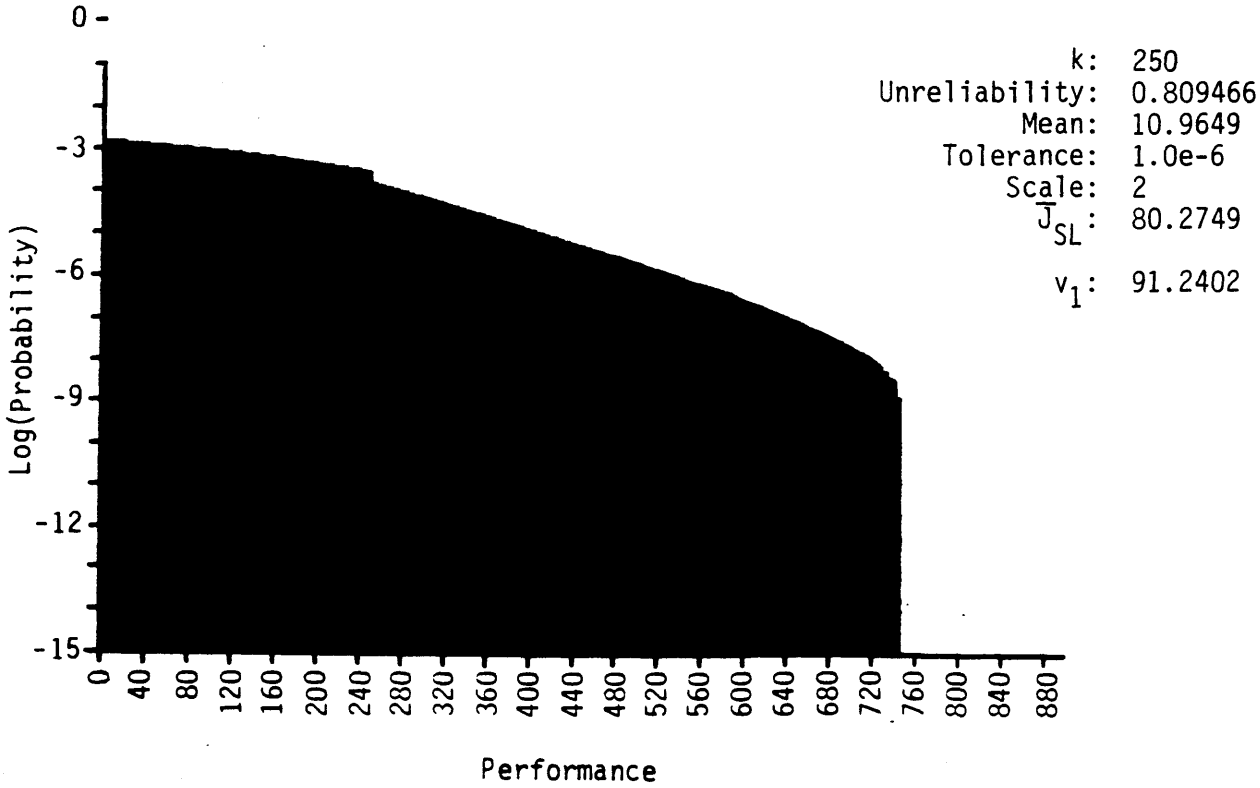


Figure 4.11. 250-Step Performance PMF for the 7-State Model

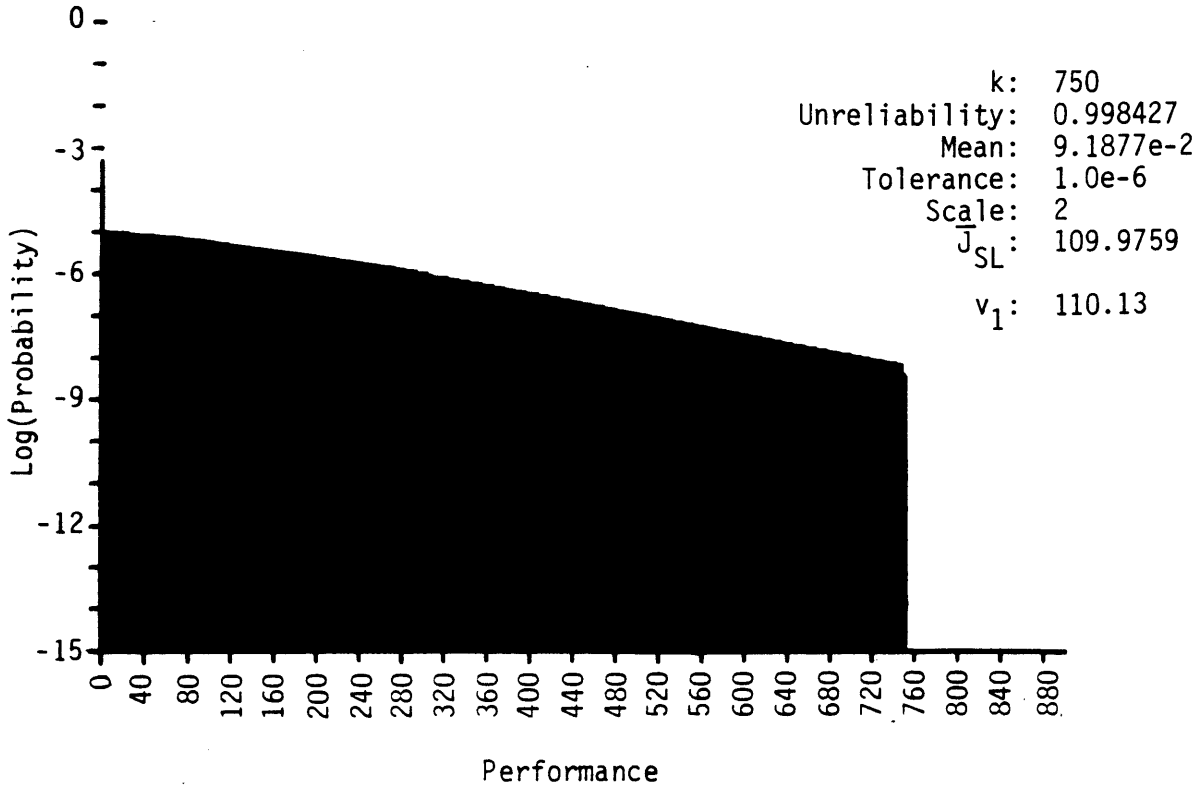


Figure 4.12. 750-Step Performance PMF for the 7-State Model

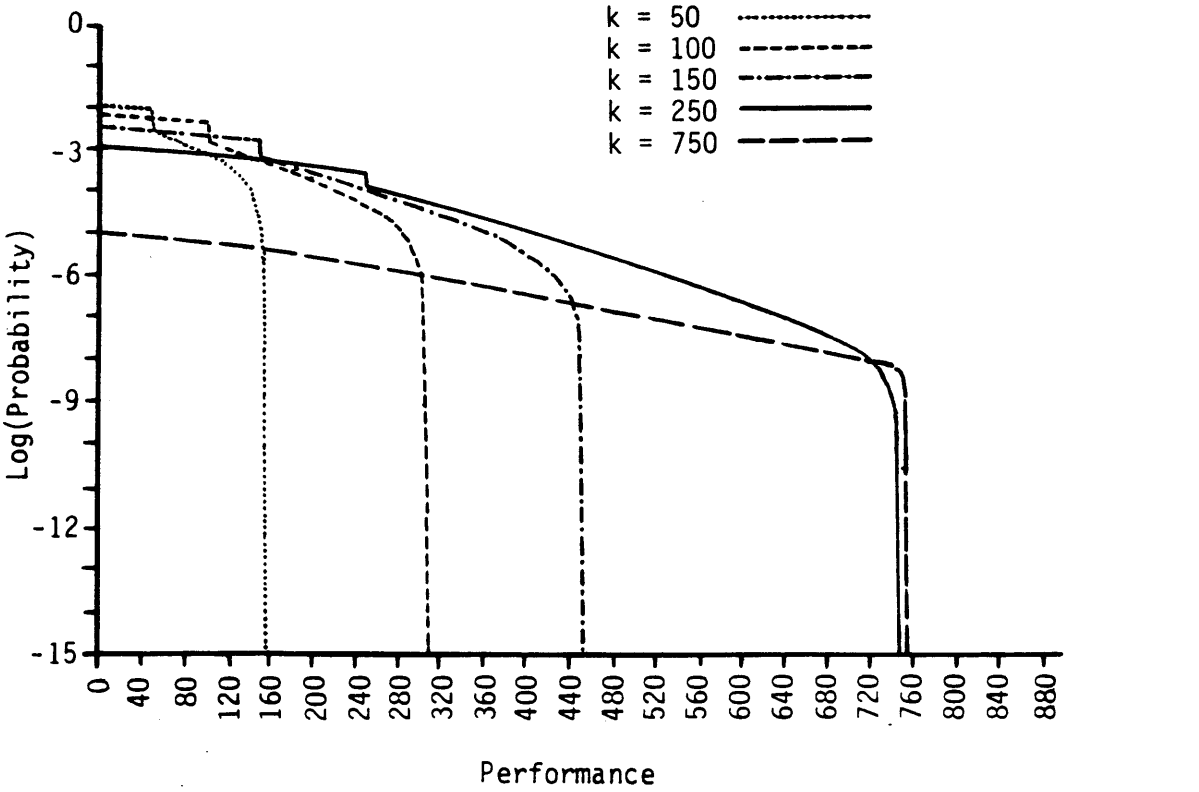


Figure 4.13. PMF Envelope Comparison for the 7-State Model

Examining a single PMF (Figure 4.8) immediately reveals a few typical PMF characteristics. First, there is an impulse at 0 on the performance scale which is almost two orders of magnitude higher than all of the other impulses. This impulse appears on all PMFs and represents the single "best" OSH which remained in the fully functional, no FDI alarm state, x_1, for the entire 50 steps. The performance of this state is 0 in the 7-state model, so this impulse lies at 0, but if the performance of this state is non-zero, this impulse and the rest of the PMF will be shifted to higher performance values. The OSH which this impulse represents will generally have the highest cumulative probability and best (lowest) cumulative performance of any OSH in the ensemble, so the impulse will always appear on the extreme left edge of the PMF.

The next dominant feature of the PMF is a flat or gently sloping plateau region about two orders of magnitude below the large impulse. This region represents the performance values for the next most likely system behavior, which is the most likely system behavior involving action by the redundancy management system. Note that the OSH represented by the large impulse involved no failures or false alarms. The performance value would therefore have been identical for a system with no RM system at all. The rest of the PMF after the impulse tells us about the performance of the system when the RM system has been forced to operate. The plateau region still involves relatively small performance values and represents OSHs which spent most of the 50 steps in "good" states (x_1 or x_3 in this case), but incurred the "expensive" transitions $x_1 \rightarrow x_2 \rightarrow x_3$ or $x_1 \rightarrow x_3$ which immediately lowered their cumulative probability by about two orders of magnitude. Depending upon the structure of the model, there can be several plateaus of varying widths and heights which can be traced to the behavior

of specific OSH subsets in the ensemble.

The final characteristic of a performance PMF is the gentle slope below the plateau which decays into a sharp roll-off. This region represents OSHs which have spent proportionally more time in states with "high" (i.e. bad) performance values. For most models, as the cumulative performance values increase beyond a certain point on the plot, the corresponding cumulative probabilities decrease very rapidly, easily reaching levels of 10^{-100} or 10^{-500}. It is safe to assume that such OSHs have negligible impact, and removing these OSHs from the ensemble to improve computational performance is the main goal of the look-ahead approximation.

Figures 4.8 - 4.12 also show the time behavior of performance PMFs. In general, as k increases, the total probability represented in the PMF decreases and moves towards higher performance values. In terms of expected performance, the mean of the PMF increases to a certain point and then decreases as the decrease in the total probability begins to dominate the spread to greater performance values. This behavior was already illustrated by the performance profile in Figure 4.6. The major features of the performance PMF also change with time. The height of the impulse at 0 decreases, and the level of the plateau decreases while its width increases. The step between the plateau and the roll-off region becomes less pronounced for larger k, and in general, all of the PMF characteristics become blurred as k increases. The PMF at mission time is the most important, however, and at k = 150, the features of the PMF are still plainly visible. Figure 4.13 shows the superimposed outlines of the PMFs to more easily illustrate their variation with time.

The next series of PMF plots, Figures 4.14 - 4.18, show the effects of

variations in the look-ahead approximation tolerance of a PMF. These plots are the 150-step PMFs of the 7 state model for tolerances of 10^{-8}, 10^{-6}, 10^{-4}, 10^{-3}, and 10^{-2}. All plots have the same horizontal scale (2 PMF impulses combine into 1 plotted impulse) which is the same scale used in Figures 4.8 - 4.12.

It should be clear that the smaller the tolerance is, the more accurate the PMF will be, because more of the "insignificant" OSHs remain in the ensemble. The "baseline" PMF is therefore the plot in Figure 4.14 with a tolerance of 10^{-8}. The accuracy of this PMF is apparent from its statistics. Its reliability agrees with the FORTRAN result to more than 5 decimal places. Furthermore, $\bar{J}_{SL}(150)$ is 43.7993 from the SCHEME profile computed at a tolerance of 10^{-10}. Adding the mean (18.9365) to $\bar{J}_{SL}(150)$ agrees with the total expected performance, $v_1(k_m)$, computed by the FORTRAN program to 5 places. By inspection, the smallest impulse appearing in this PMF (on the right edge) that was not removed by the approximation has a probability of about 10^{-10}, which is insignificant.

In Figure 4.15, the tolerance has been raised to 10^{-6}. Note that the statistics, to the accuracy they are represented, are exactly the same as those of 4.14, even though the smallest impulse represented now has a slightly greater probability, about 10^{-8}. In 4.16, the tolerance is 10^{-4}, and now the statistics of the PMF have changed very slightly - perhaps not by what would be considered a significant amount, but some accuracy has been lost. As the tolerance is raised to 10^{-3} and then 10^{-2} in Figures 4.17 and 4.18, the shape of the PMF and its statistics change much more drastically, such that the PMF in Figure 4.18 is useless as an analytic tool. Note, however, that for good tolerances like 10^{-6} and smaller, the PMF has a maximum value of about 450 for 150 time steps. This indicates

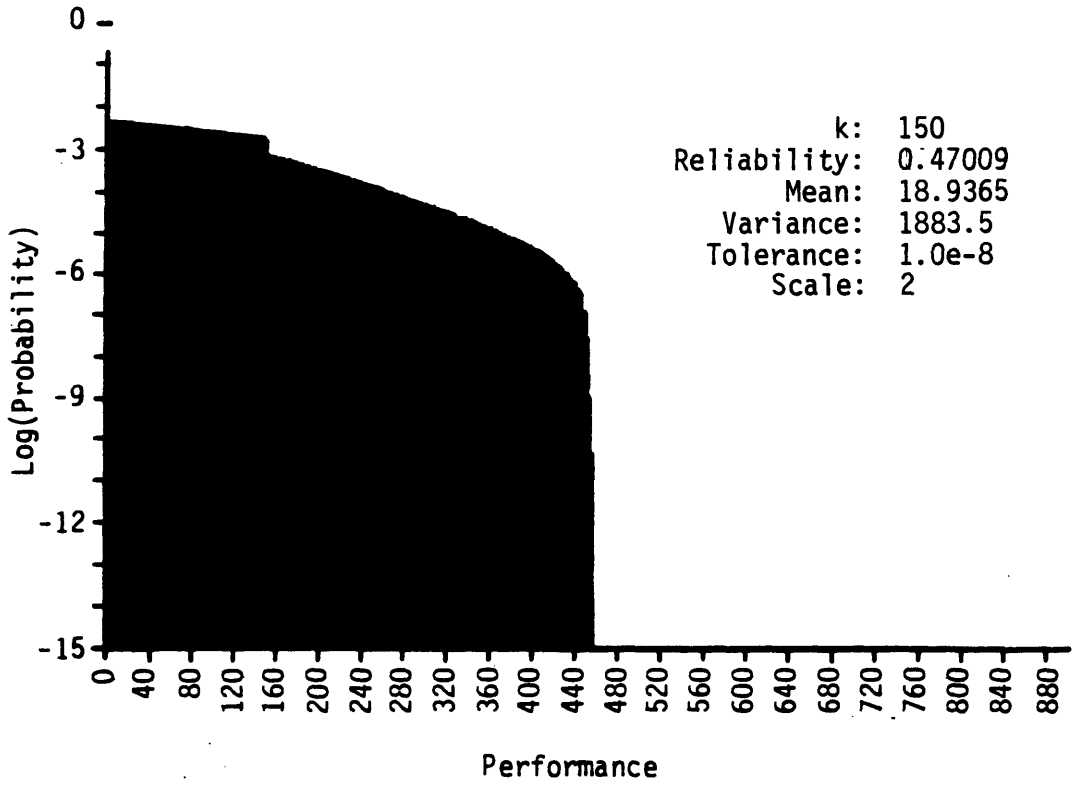


Figure 4.14. 150-Step Performance PMF for a Tolerance of 10^{-8}

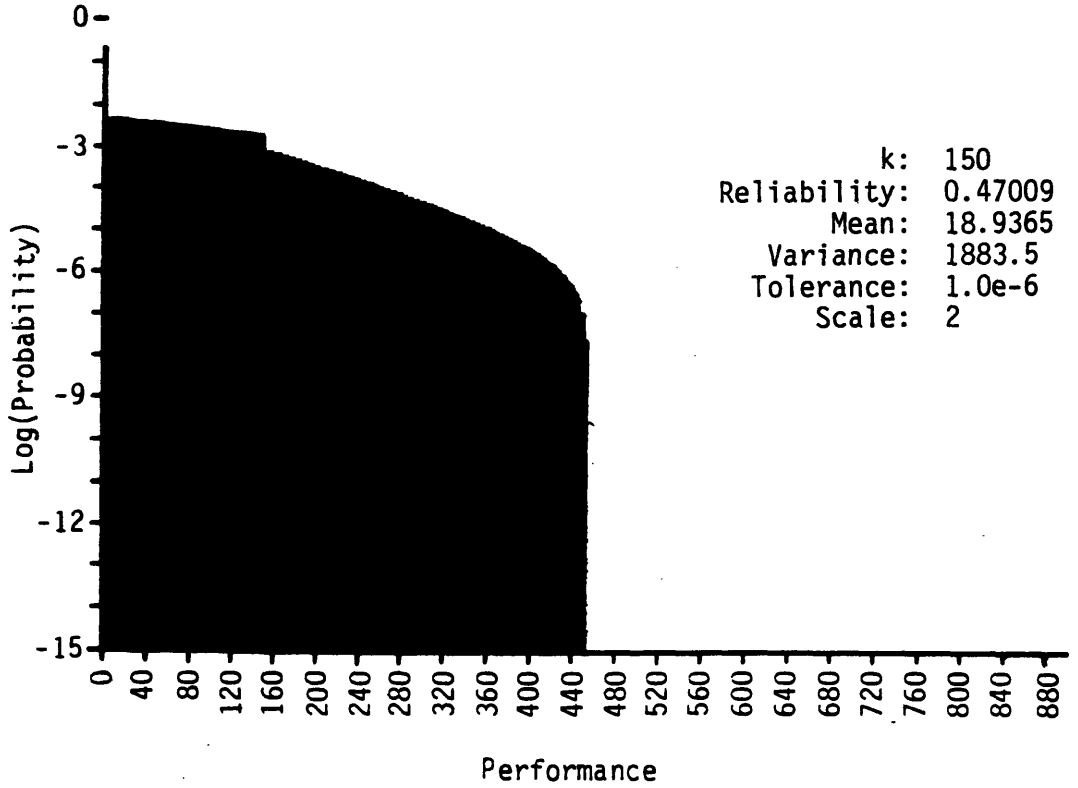


Figure 4.15. 150-Step Performance PMF for a Tolerance of 10^{-6}

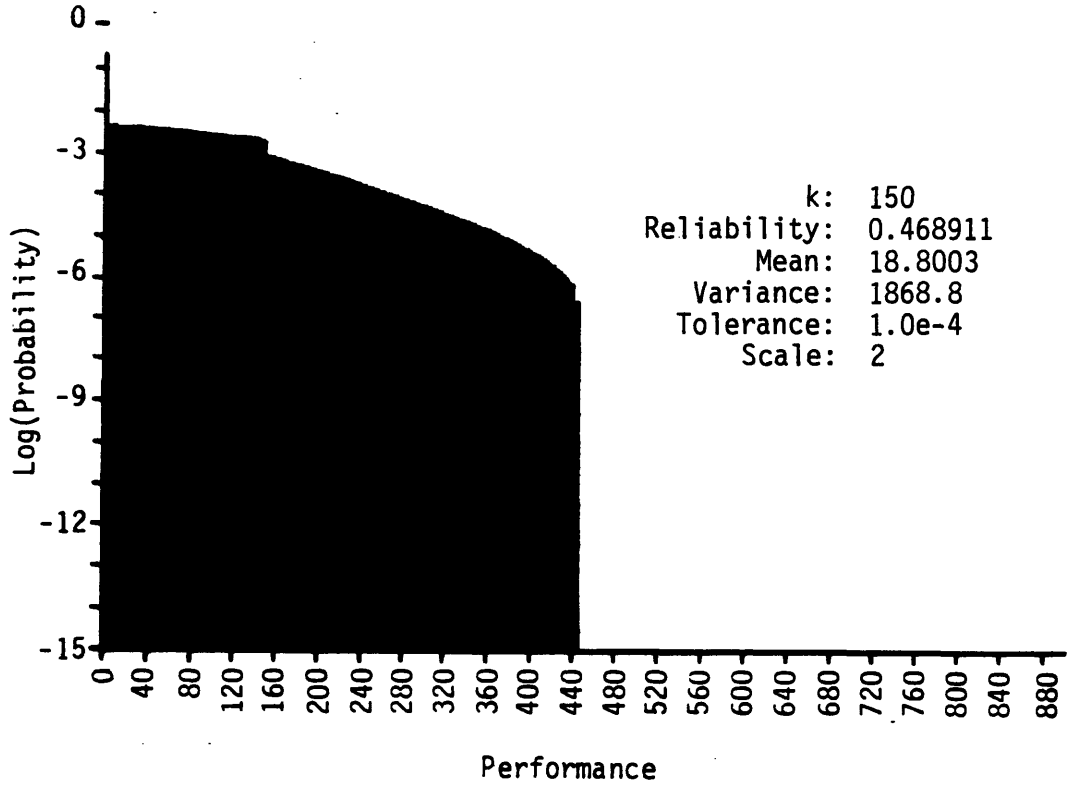


Figure 4.16. 150-Step Performance PMF for a Tolerance of 10^{-4}

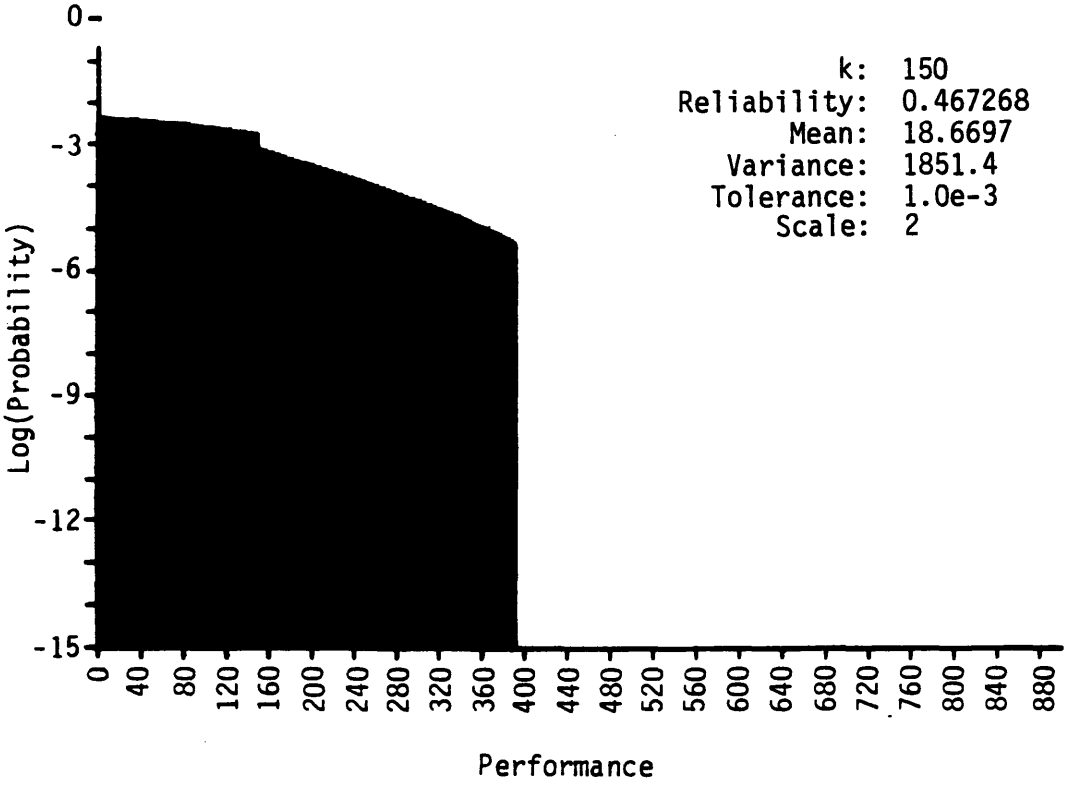


Figure 4.17. 150-Step Performance PMF for a Tolerance of 10^{-3}

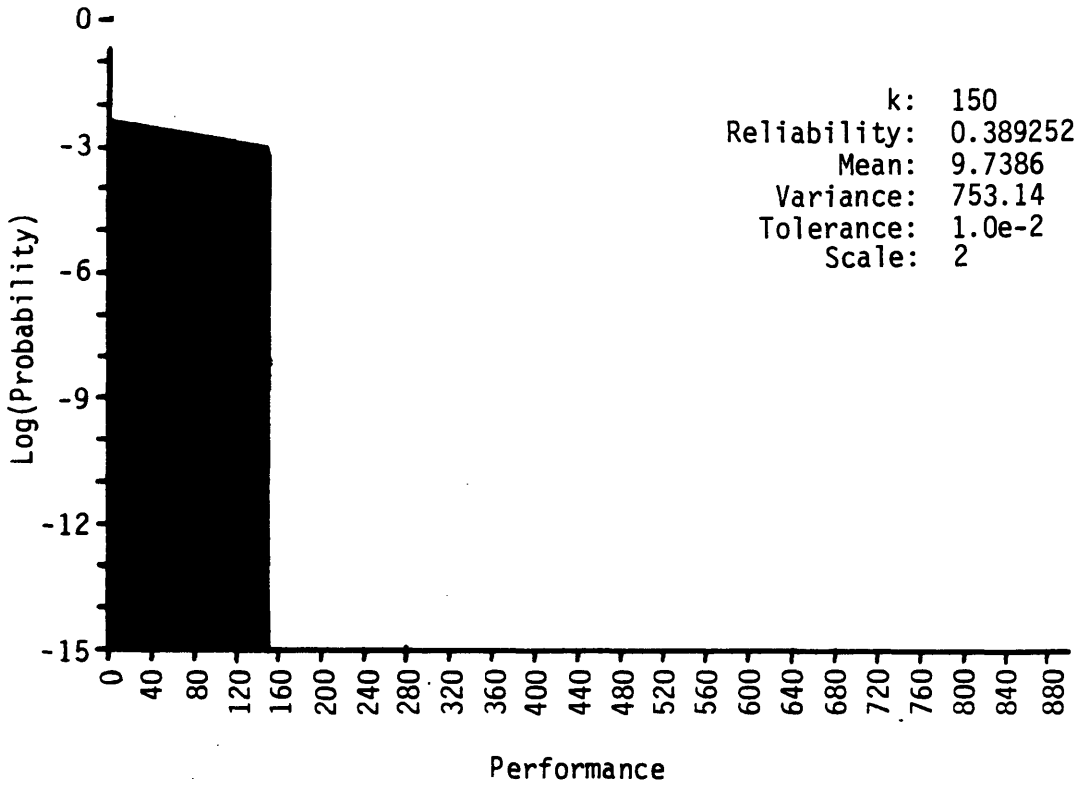


Figure 4.18. 150-Step Performance PMF for a Tolerance of 10^{-2}

that the OSHE has grown at an average rate of 3 OSHs per step, which is smaller than the upper bound of 5 OSHs per step predicted by the largest performance value in the model. Also note the typical sizes of the variances of the PMFs. Variances which are so large relative to the mean reinforce the idea that realistic performance evaluation cannot be solely based on an expected performance profile of a system. The performance PMF should be used first because it provides much more information about a system and is less likely to be misleading.

Selecting a "good" tolerance to use during analysis is a problem of trade-offs. If the tolerance is too small, the computer will need extra memory space and more computation time to store and manipulate the additional v-transforms of insignificant OSHs. However, a tolerance which is too large can significantly alter the PMF, reducing it to either nothing

or just the largest (leftmost) impulse, and simultaneously destroying the statistics. Therefore, selecting a tolerance to use for the look-ahead approximation is a compromise between accuracy, computer memory usage, and computation time. There should be a tolerance which allows a reasonable PMF to be computed for $k = k_m$ in most problems. If not, more powerful computational resources must be found, or else the problem must be recast with less severe computational requirements.

Earlier, it was claimed that $v_1(k_m)$ is a good guide for selecting a tolerance. For the 7-state model, $v_1(150)$ is 110, or about 10^2. The PMFs in Figure 4.14 - 4.18 used tolerances that were 10, 8, 6, 5, and 4 orders of magnitude smaller than $v_1(150)$. For this problem, results were still very accurate for a tolerance 6 orders of magnitude below $v_1(150)$, but results suffered for tolerances larger than this. After gaining experience with the other models to be presented shortly, a tolerance 6 orders of magnitude below $v_1(k_m)$ worked well as a rule of thumb. If it is computationally possible, it is reasonable to go as far as 9 or 10 decades down from $v_1(k_m)$ in choosing the tolerance, but using tolerances smaller than 10 decades down for general performance evaluation problems is wasteful and probably will not improve the results.

4.4.2 Modeling Approximations (The 50- and 10-State Models)

The 50-state model described in Section 4.3 was a severe test of the SCHEME system. The FORTRAN program had no difficulty generating results some of which can be found in Appendix C. FORTRAN provided $v(7200)$ ($k_m = 7200$) which was used by the SCHEME system. The first element of $v(7200)$ is 101,296., or about 10^5. In light of this, two tolerances were used with the SCHEME system: 10^{-4} (9 decades down) and 0.1 (6 decades

down) according to the framework for selecting tolerances described above. Unfortunately, this model quickly reached the memory limit of the HP9826. The PMF for k = 111 was the greatest that could be calculated for the tolerance 10^{-4}, and the PMF for k = 376 was the best possible for a tolerance of 0.1. These limits were reached so soon because the HP9826 completely relies on physical RAM for memory and has no virtual memory capability. On a larger computer with virtual memory, there is little doubt that this problem could be analyzed for much larger values of k.

Rather than abandoning the 50-state model, it can be used to demonstrate an important application of performance evaluation: Model simplification. In the past it has been a common practice during reliability analyses to either ignore or aggregate many states of a large Markov model in order to make the model analytically tractable. Selecting the states to ignore or aggregate has generally been based more on intuition than on solid indications of how using a reduced order model will affect the results of the analysis. Examining the "modal coefficients" of a Markov model calculated during performance evaluation can partially rectify this problem.

In the FORTRAN program, the value of $v_1(k)$ is the only value calculated in order to generate a total expected performance profile. To calculate $v_1(k)$, the expression derived in Chapter Three is used directly:

$$v(k) = rPV \; \mathrm{diag}\left[\frac{\lambda_i^k - 1}{\lambda_i - 1}\right] W$$

Since r is a row vector and P and V are square matrices, rPV will also be a row vector, and rPV multiplied by the diagonal matrix of modal geometric sums will also be a row vector. To find $v_1(k)$, the inner product of the row vector

$$rPV \; diag\left[\frac{\lambda_i^k - 1}{\lambda_i - 1}\right]$$

and the first column of W is all that needs to be computed. The expression for $v_1(k)$ will take the following form:

$$v_1(k) = a_1\left[\frac{\lambda_1^k - 1}{\lambda_1 - 1}\right] + a_2\left[\frac{\lambda_2^k - 1}{\lambda_2 - 1}\right] + \ldots + a_{S-1}\left[\frac{\lambda_{S-1}^k - 1}{\lambda_{S-1} - 1}\right]$$

(Recall that $\lambda_S = 1$ and is left out of the problem by partitioning P.) The coefficients "a_n" are the modal coefficients. If $[rPV]_i$ is the ith element of the row vector rPV and W_{i1} is the ith element in the first column of W, then

$$a_i = [rPV]_i W_{i1}$$

The modal coefficients indicate which Markov states contribute the greatest amount of total expected performance in $v_1(k)$. Therefore, these coefficients serve as a guideline for which model states are the most important in terms of performance. The modal coefficients of the 50-state model are in an output listing in Appendix C.

Among the 49 modal coefficients, there are 9 (highlighted by boxes) which are from 3 to 5 orders of magnitude larger than the other 40. For this system, it might be expected that these 9 coefficients correspond to the states with the 9 largest (> 0.9995) self-loop probabilities or eigenvalues. Though specification of the entire 50-state model had to be automated, the origins of these 9 dominant states are easy to trace. In the 8-state subsystem models, 3 states have the largest self-loop probabilities for both the sensors and the actuators: x_1, x_4, and x_7, which correspond to component configurations 4/0, 3/0, and 2/0, respectively. When the actuator and sensor subsystems are combined, the 9 dominant states of the 50-state model are the various combinations of each

of the the 3 dominant sensor states with each of the 3 dominant actuator states. Note that in this case, the dominant states are <u>not</u> the states with the largest performance values.

Knowing which states dominate the 8-state and 50-state models, the greatly simplified 4-state models of the sensor and actuator subsystems in Figure 4.19 were constructed. Half of the states and the complicated transition structure of each subsystem have been ignored. The upper three states in each model correspond to the states x_1, x_4, and x_7 for each subsystem, and take their self-loop probabilities and performance values from these states. When the two 4-state models are combined (once again assuming independent subsystems), a 10-state model results which should reasonably approximate the original 50-state model from a performance evaluation standpoint. A listing of the eigenvalues and modal coefficients for the 10-state model is in Appendix C.

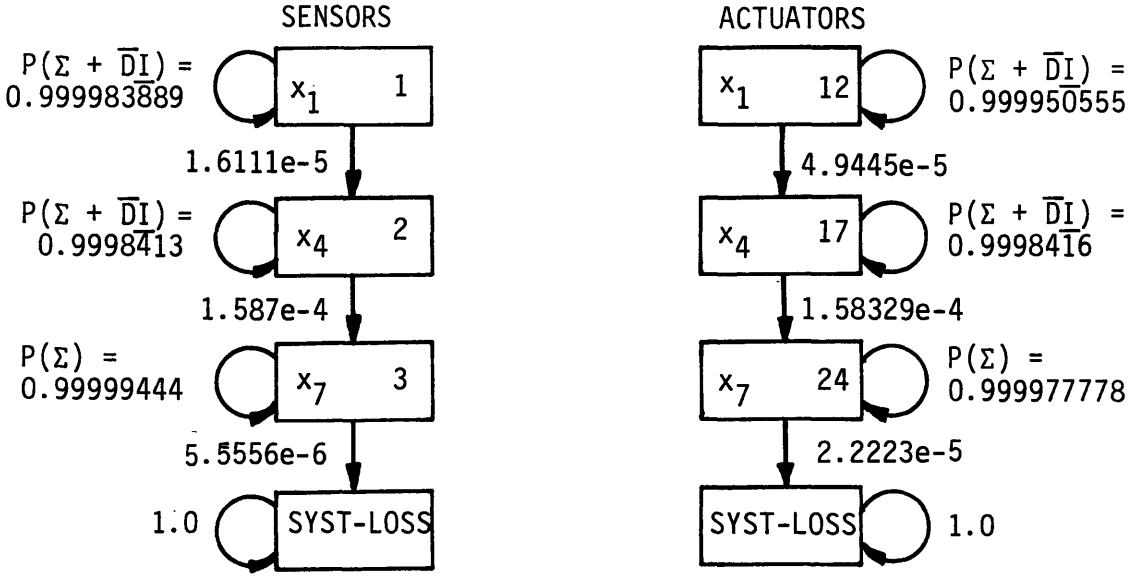


Figure 4.19. Approximate Subsystem Models

Figure 4.20 is a comparison of the total expected performance profiles

of the 50-state and 10-state models. The profiles match relatively closely, developing significant discrepancies only for times much larger than mission time. A magnification of the same plot is shown in Figure 4.21, enlarged so that the behavior for times near k_m can be seen. Clearly, the two models match very closely in terms of expected performance.

Using the SCHEME procedures which calculate performance PMFs directly without accumulating statistics (Appendix B, "FAST.SCM"), the 350-step PMFs for the 10- and 50-state models were calculated using a tolerance of 0.1 (Figures 4.22 and 4.23). The PMFs are scaled by 15 to produce these plots, which are very similar visually. The mean values match reasonably well. From the total performance profile, the expected performance $v_1(350)$ is approximately 4560. At $k = 350$, a negligible amount of expected performance has reached the system-loss state because 350 is so small compared to 2.5×10^5, the approximate steady-state time. Therefore, $v_1(350)$ approximately equals the mean values of both PMFs.

The PMFs differ largely in their reliabilities and variances. Many low probability transitions in the 50-state model are eliminated in the 10-state model. The elimination of these extra paths to the system-loss state leads to a significantly higher predicted reliability for the 10-state model. The 50-state model also has several extra impulses beyond the 6300 performance figure which is the largest 10-state model performance value. Though these impulses affect the mean very little, they contribute more significantly to the variance.

In this instance, the 10-state model proves to be a useful approximation to the 50-state model. This approximation may seem obvious because both subsystems have the same easily identifiable structure, but

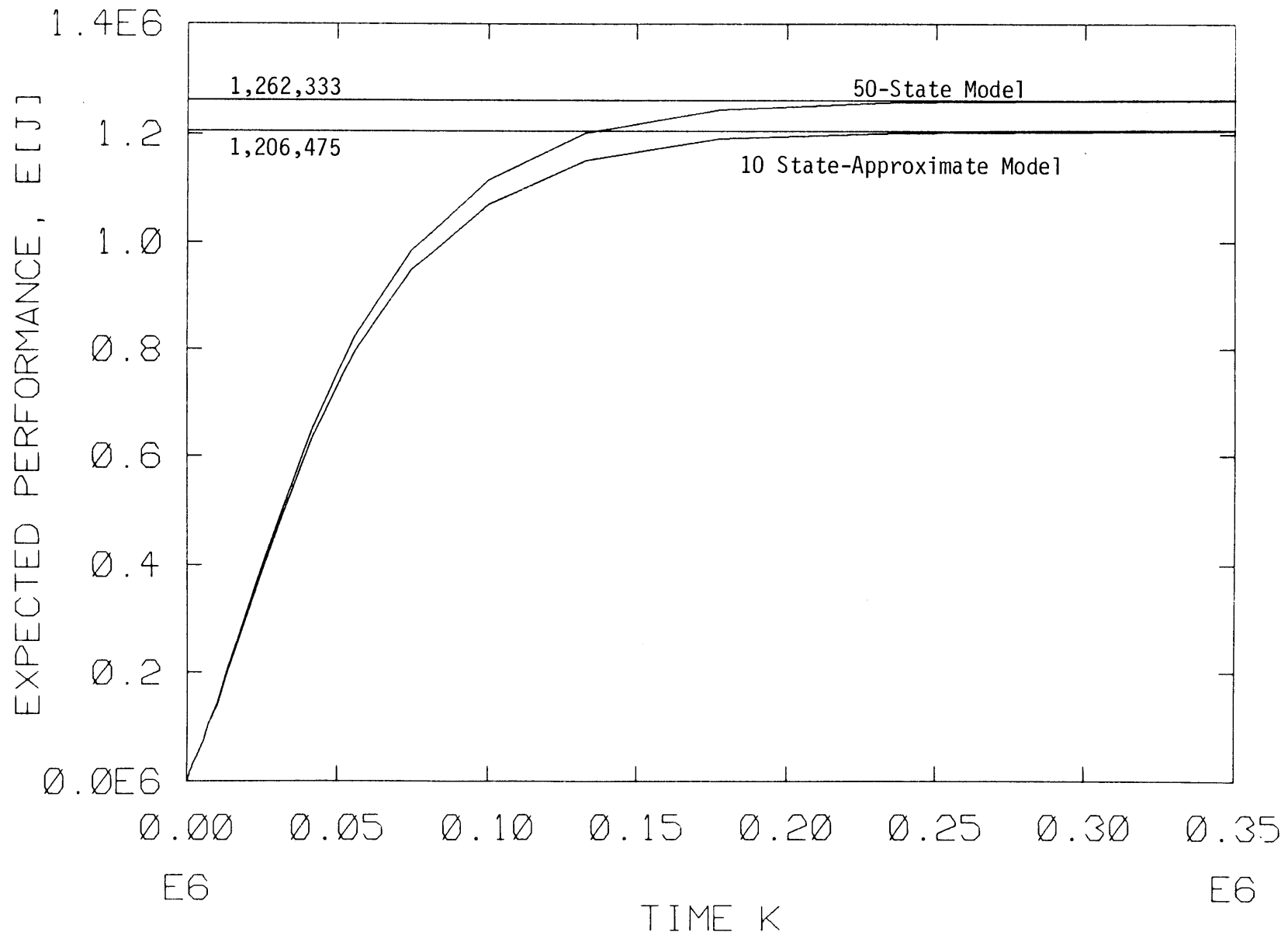


Figure 4.20. Total Expected Performance Profiles for the 10- and 50-State Models

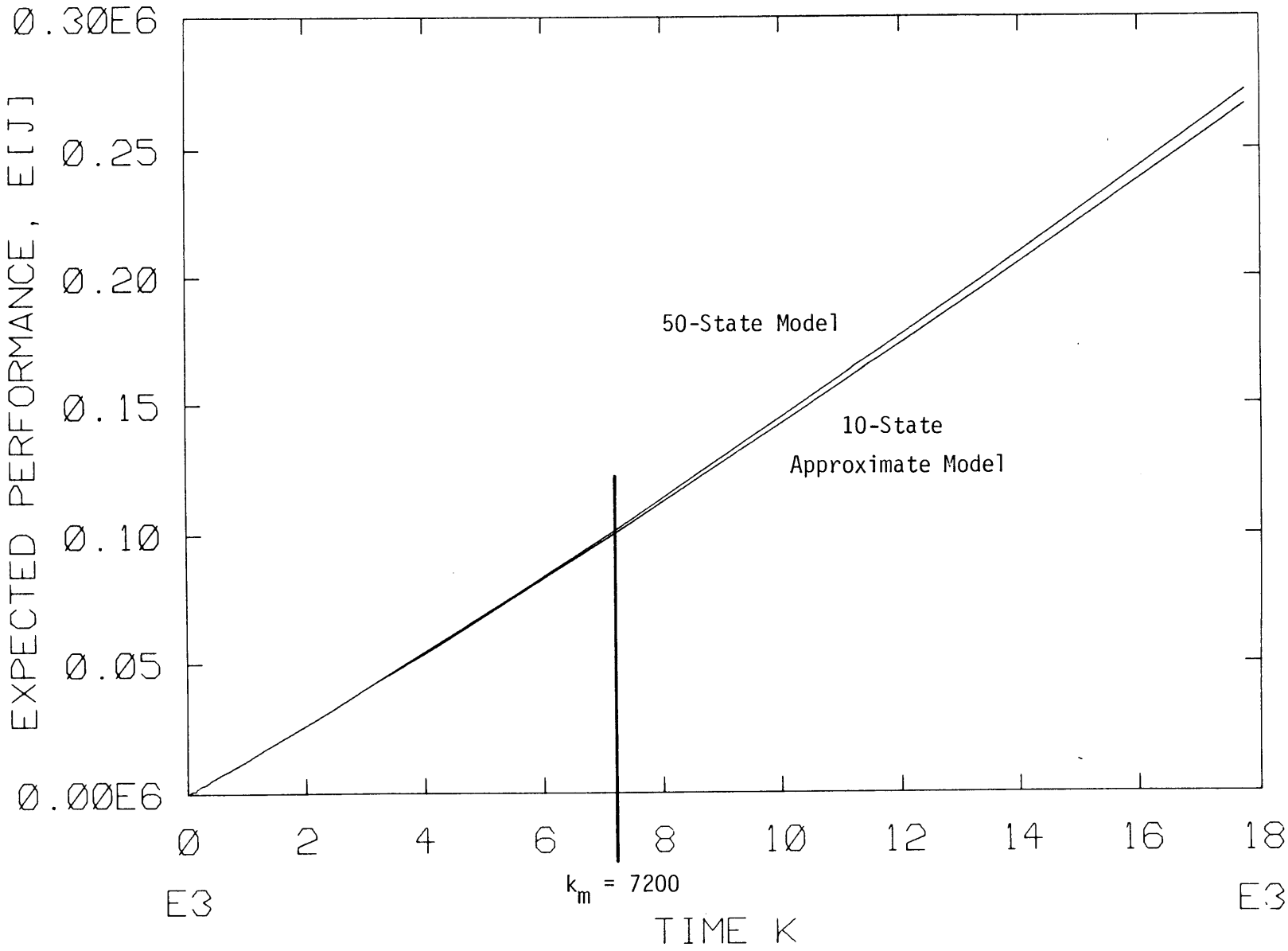


Figure 4.21. Magnification of Figure 4.20

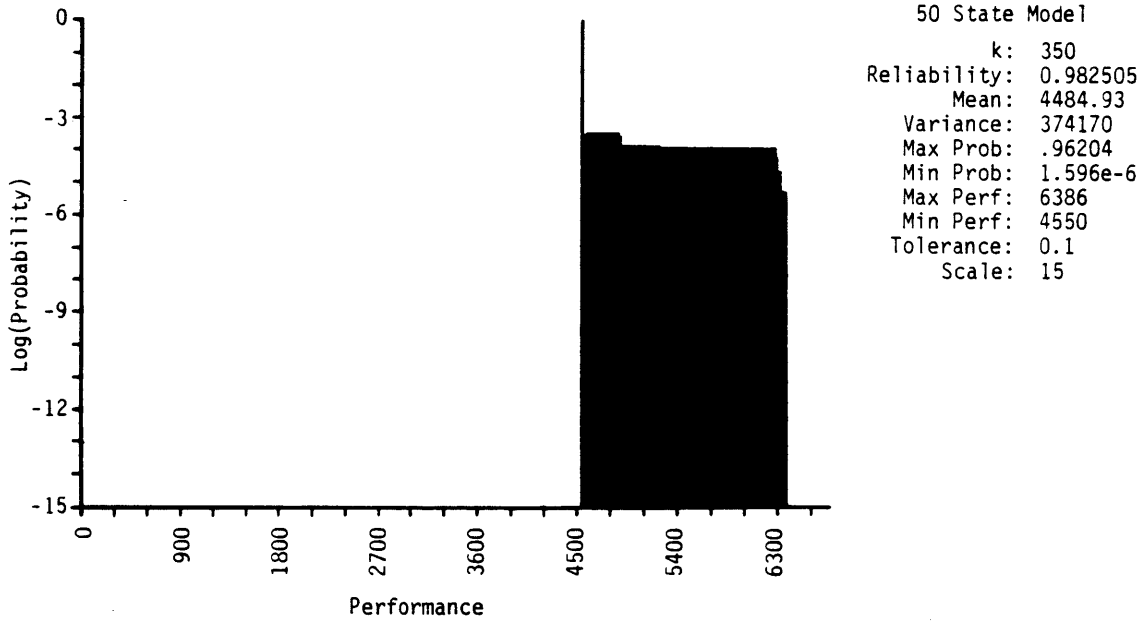


Figure 4.22. 350-Step Performance PMF for the 50-State Model

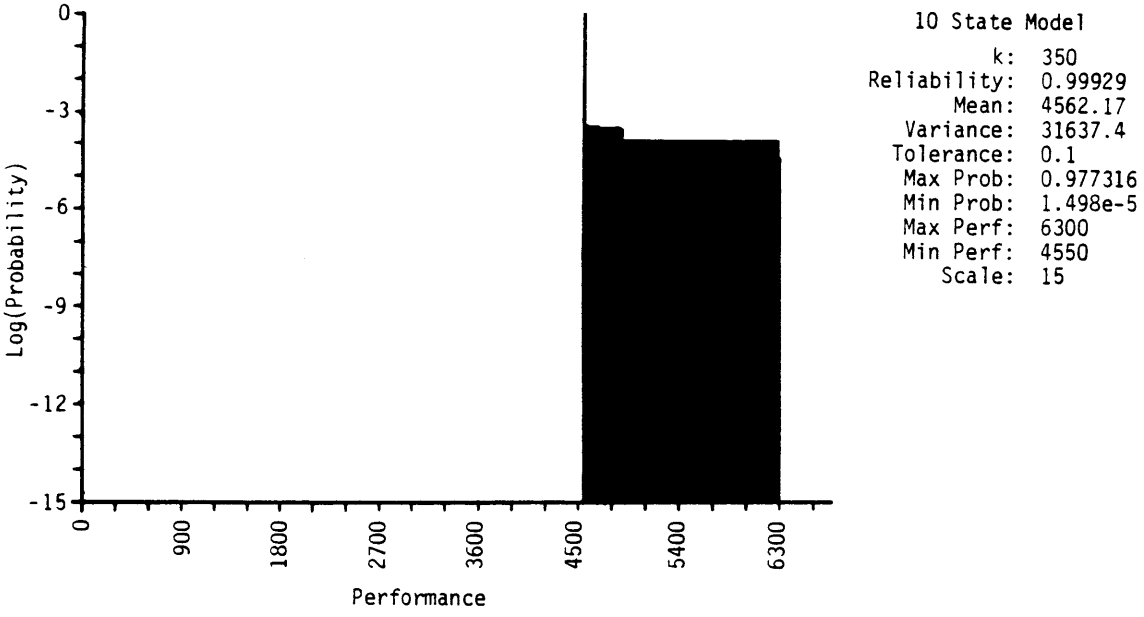


Figure 4.23. 350-Step Performance PMF for the 10-State Model

examining the modal coefficients and eigenvalues may be useful in other more complicated systems with dominant states that are difficult to identify. Regardless of how an approximate model is constructed, performance evaluation of a complete model will still be helpful in determining whether the approximations are valid.

4.4.3 Parameter Variation Analysis

The most promising application of the performance evaluation methods is as a design tool for redundancy management systems. Performance PMFs and expected performance profiles permit the comparison of RM systems that use different components, FDI strategies, reconfiguration algorithms, plant dynamics, or any other system factor that can be modeled. Using such comparisons, the engineer can make sound decisions during the RM system design process.

To demonstrate the use of performance evaluation in an RM design context, the 8-state sensor subsystem model described in Section 4.3 was reconfigured for variations in two key parameters: The mean-time-to-failure (MTTF) of the components, and the false alarm probabilities, P_{FA}, of the FDI system for states 4/0 (x_1) and 3/0 (x_4). The table below summarizes the six cases that were analyzed. All cases have the same performance structure as the sensor subsystem described in Section 4.3, and case 1 has the same parameters which were used for that subsystem. More detailed information about the models for cases 2 through 6 is presented in Appendix A.

The first phase of performance evaluation is the calculation of total expected performance profiles and the $v(k_m)$ vectors for each case. Short listings of the FORTRAN output for each case are in Appendix C. Figure

	MTTF	P_{FA} 4/0	P_{FA} 3/0
CASE 1	100	0.001	0.005
CASE 2	100	0.005	0.01
CASE 3	100	0.05	0.1
CASE 4	10	0.001	0.005
CASE 5	10	0.005	0.01
CASE 6	10	0.05	0.1

4.24 shows the total expected performance profiles for all six cases for times long enough for all systems to reach steady-state. The systems with the higher component MTTFs of 100 hours take longer to reach steady-state than the systems with MTTFs of only 10 hours. Component MTTFs are the basis for most of the self-loop probability of the dominant model states. Note also that the systems with long MTTFs reach much higher expected performance values. This behavior is due to the fact that the OSHs in these cases spend more time in the "operational" states with non-zero performance values before ultimately moving to the system-loss state.

An interesting feature of the curves for cases 2 and 3 is that they cross in the area of k = 200,000. Though this k is much larger than mission time (7200), prior to k = 200,000, case 2 would be the more desirable system based on total expected performance alone. After k = 200,000, case 3 is more desirable. Though this conclusion is not very meaningful due to the magnitude of k, it is the first indication that the steady-state behavior of the systems does not accurately reflect their transient performance.

The mission time for all six cases is 7200 steps. Figure 4.24 is magnified in Figure 4.25 to reveal the behavior of the profiles in the area

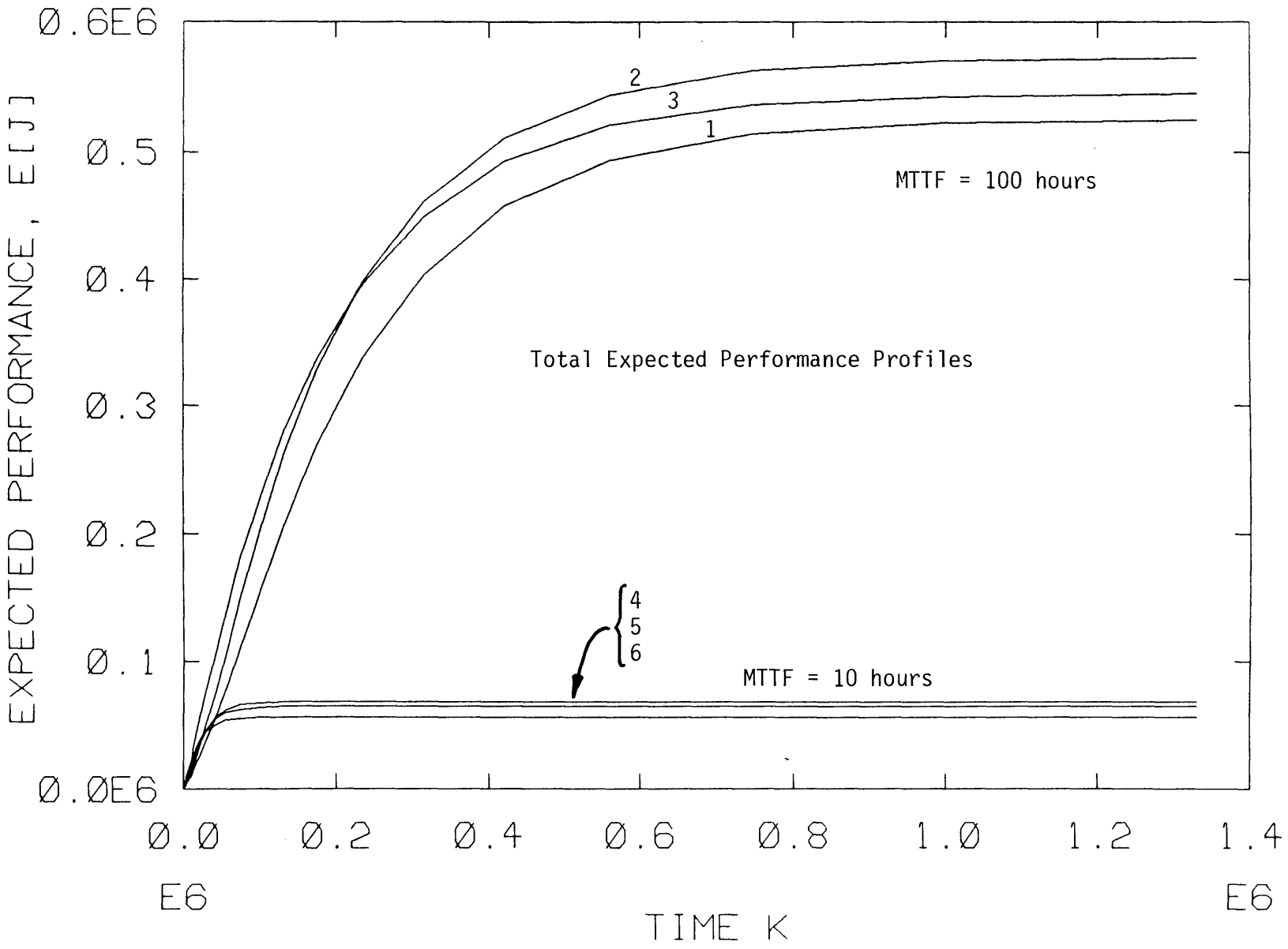


Figure 4.24. Total Expected Performance Profiles for 6 Cases

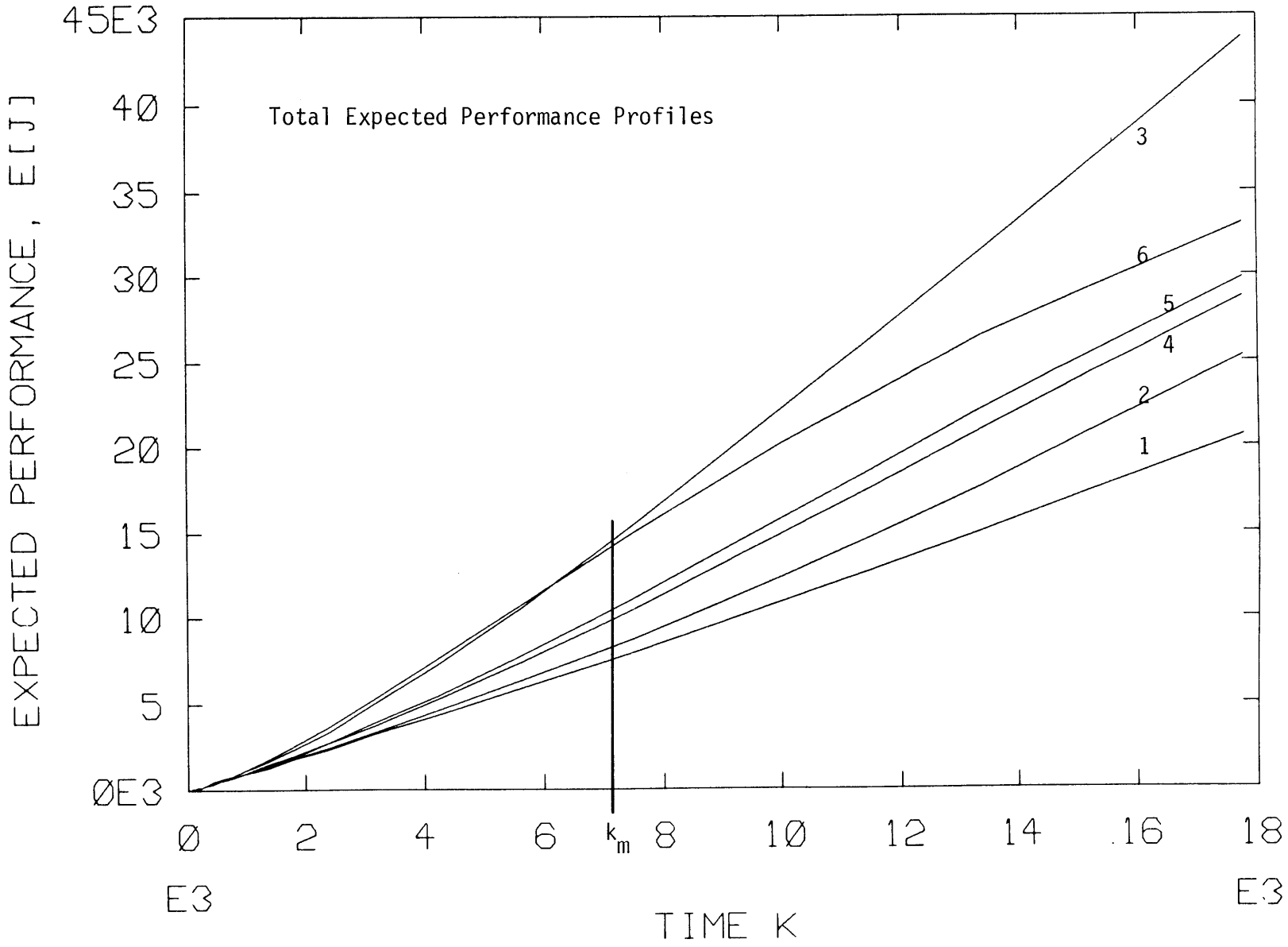


Figure 4.25. Magnification of Figure 4.24

of k = 7200. As Figures 4.24 and 4.25 demonstrate, the differences between the short-term and long-term behavior of the systems can be quite dramatic. Recall that profile curves 1, 2, and 3 have the long rise time and must eventually rise above curves 4, 5, and 6. Yet in Figure 4.25, cases 1 and 2 have the lowest profile curves. The lowest curves have better performance because the performance values are larger for model states with poor performance. Cases 4 and 5 are the next best after 1 and 2, and case 3 is the worst, though it crosses curve 6 only shortly before k_m. Since curves 3 and 6 are so close at k , one could conclude that using 100 hour MTTF components rather than the 10 hour MTTF components is not worth the additional expense unless the FDI system can be improved. Since the total expected performance profiles do not reveal the expected performance of the PMF and the expected performance of the SLOSHs, we move to the second phase of performance evaluation.

Using the $v(k_m)$ vectors calculated by the FORTRAN program, 1000-step performance PMFs for the 6 cases were calculated with the SCHEME system (Figures 4.26 - 4.31). Limits of the microcomputer prevented 7200-step PMFs from being calculated, but once again, this would not be a problem on a mainframe computer with virtual memory. The first element of $v(k_m)$ was on the order of 10^4 in all 6 cases, so a conservative tolerance of 10^{-4} -- 8 decades down from $v_1(k_m)$ -- was used for the look-ahead approximation to insure accurate results. With this tolerance, OSHE growth was about 2 OSHs per step. The plots have a scale factor of 7, i.e. 7 PMF impulses were added to generate each plotted impulse. Each plot also includes a variety of other information. The "Max Prob" is the height of the large impulse on the left side of each plot, while "Min Prob" is the height of the smallest impulse, showing that the look-ahead approximation culled all impulses with

probabilities less than "Min Prob." "Max Perf" and "Min Perf" show the range of performance values represented by the PMF, and for these models, the performance scale is proportional to the variance of the sensor measurement noise (see Appendix A).

The PMFs amplify the information provided by the expected performance profiles. The two plateaus of case 1 are lower than the corresponding plateaus or roll-off regions in any of the other cases, showing that case 1 would be considered the best system, as our intuition would indicate. The plateaus for cases 3 and 6 are missing entirely, and the level of the flat region prior to roll-off is almost one order of magnitude higher than any of the other plots. Note also the mean values and variances of each plot. In order of increasing mean values and variances, the PMFs ordered by case number would be 1, 2, 4, 5, 3, and 6. Comparing cases 1, 2, and 3, and 4, 5, and 6 as separate groups of three, we see that the height of the largest impulse (Max Prob) decreases slightly between the first two cases in each group and is drastically lower in the third case (3 and 6). Because the false alarm probabilities grow progressively larger for each of these groups, the probability that the system will remain in the state x_1 for the entire 1000 steps decreases.

Two examples will show how the performance PMFs function in RM system design. Suppose that beginning with the system in case 6, we wanted to know whether it would be better to use the more reliable 100 hour MTTF components, as in case 3, or to improve the FDI system, as in case 5. The superimposed outlines of the PMFs for cases 3, 5, and 6 are presented in Figure 4.32. Clearly, case 3 is only slightly better than case 6, whereas case 5 is much better - including an order of magnitude decrease in the roll-off region probability. Therefore, it is better in this example to

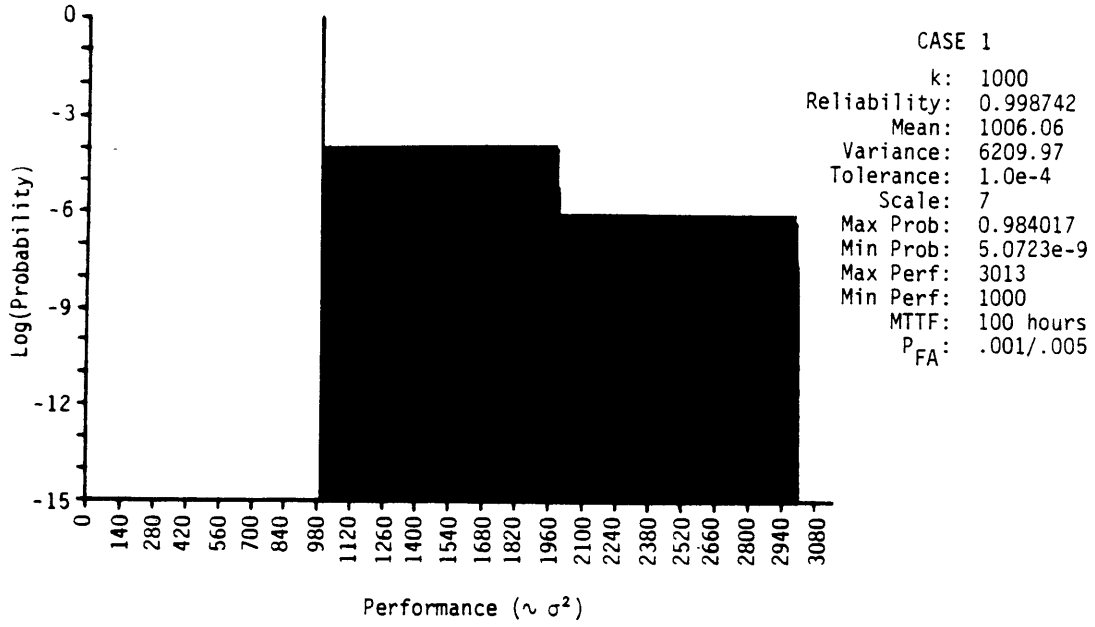


Figure 4.26. 1000-Step Performance PMF for Case 1

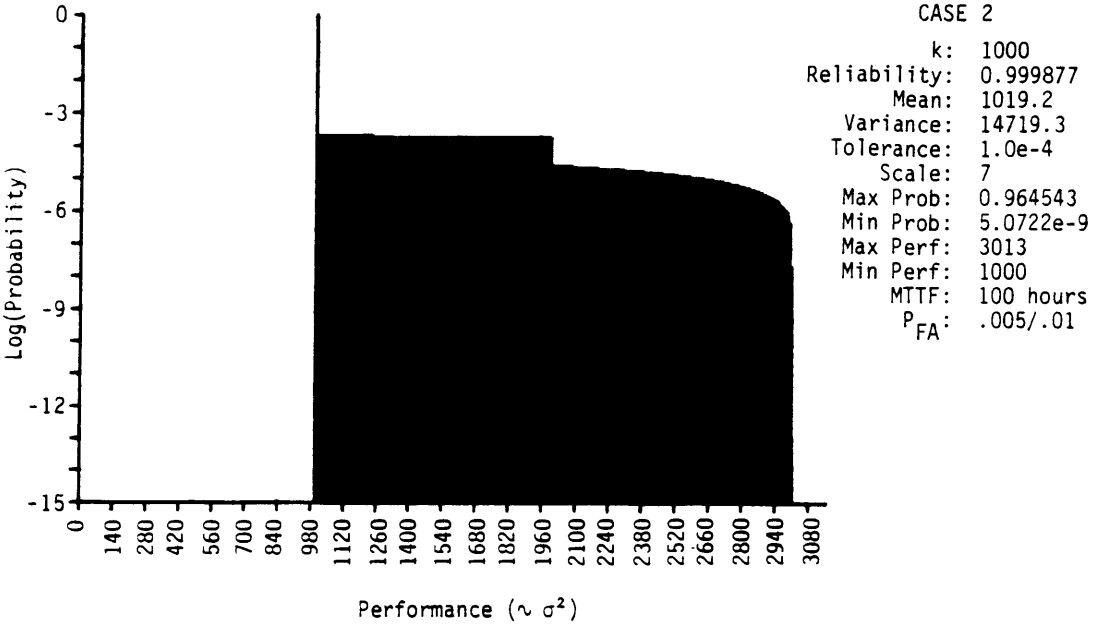


Figure 4.27. 1000-Step Performance PMF for Case 2

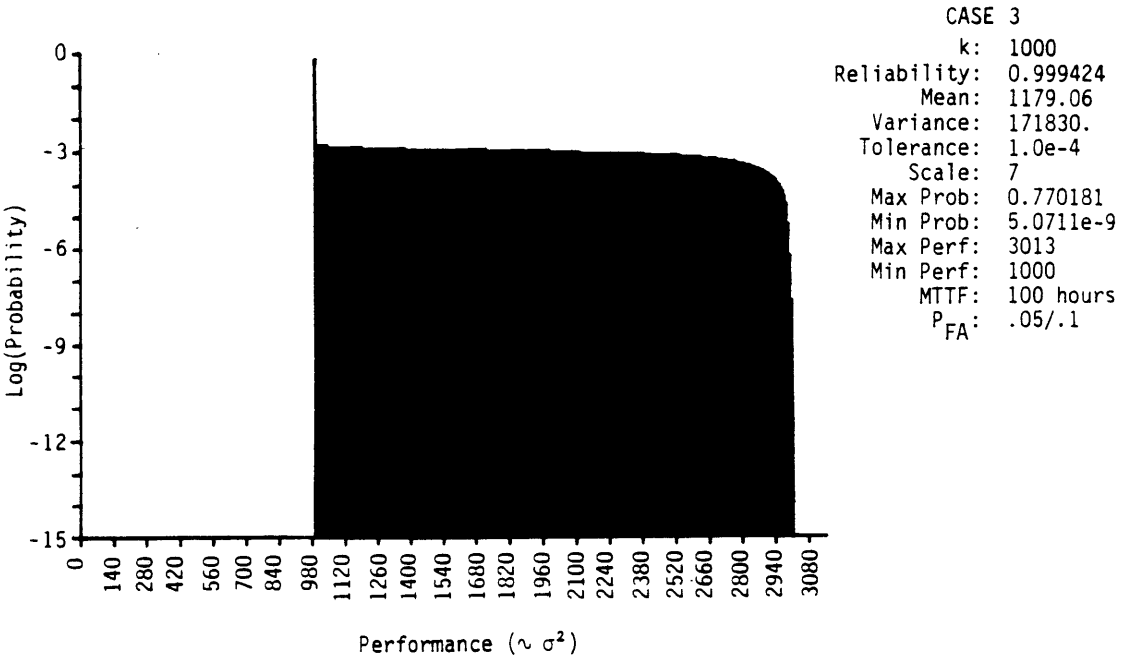


Figure 4.28. 1000-Step Performance PMF for Case 3

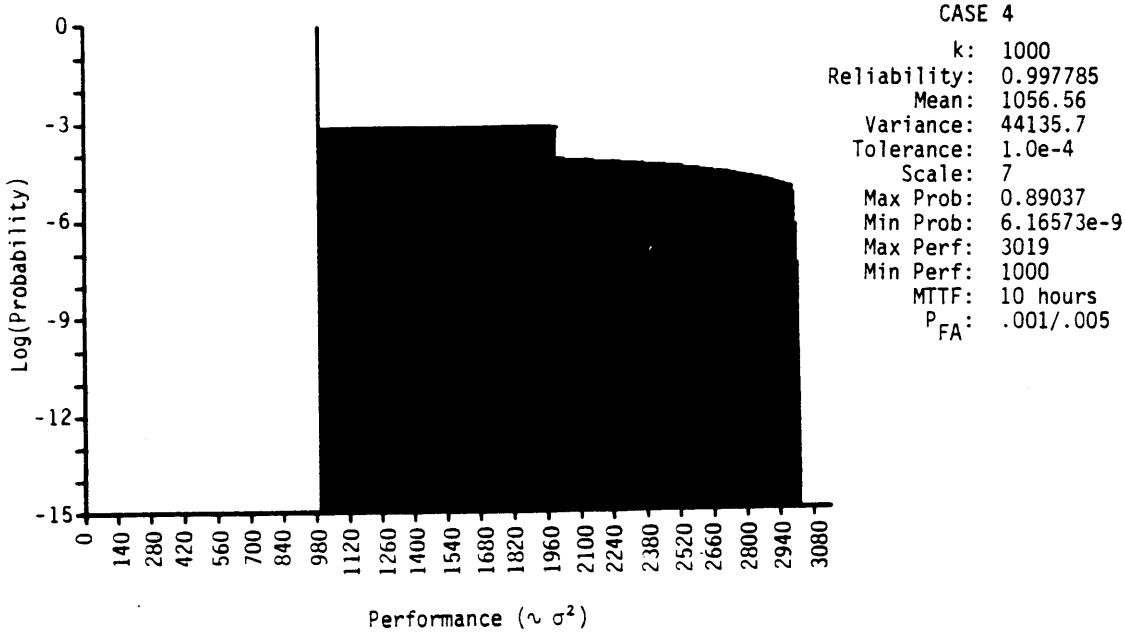


Figure 4.29. 1000-Step Performance PMF for Case 4

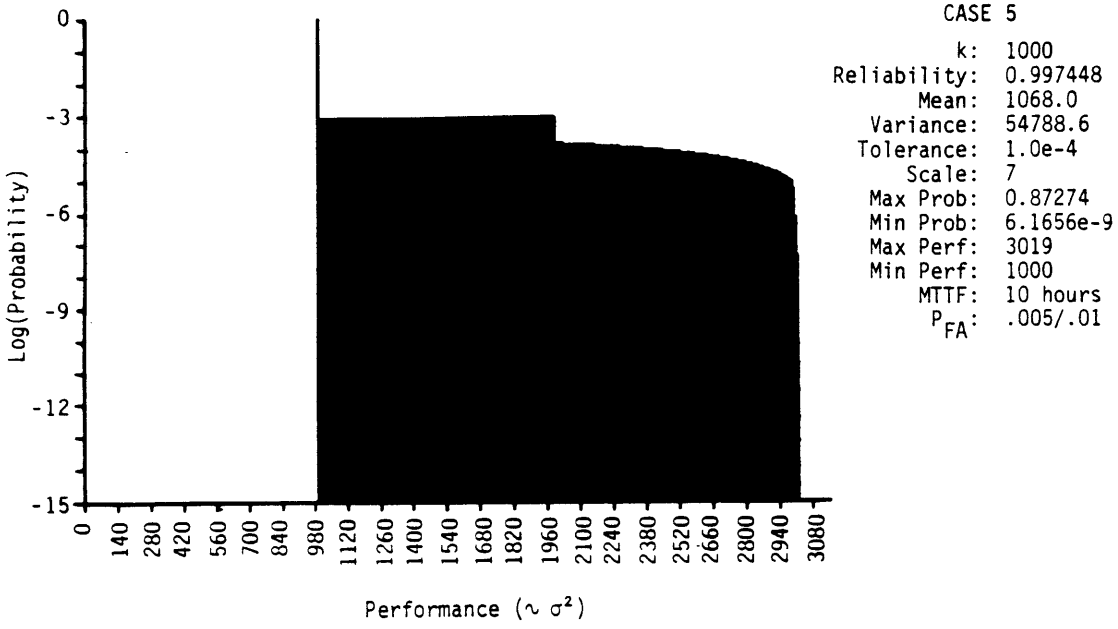


Figure 4.30. 1000-Step Performance PMF for Case 5

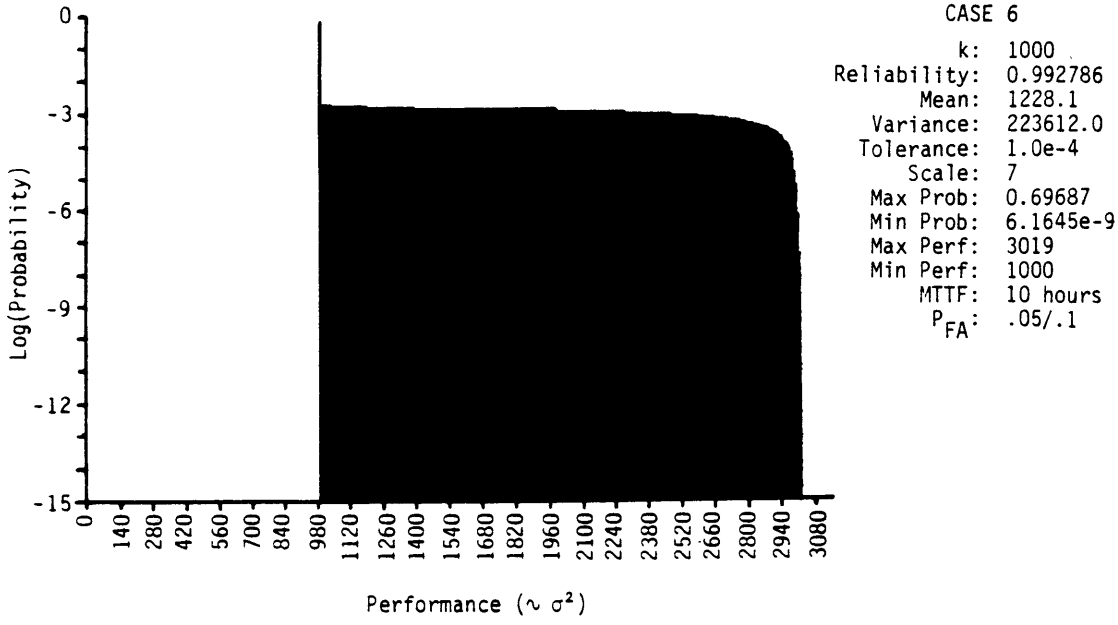


Figure 4.31. 1000-Step Performance PMF for Case 6

work on improving the FDI system rather than to waste resources on better components.

For the second example, suppose that the baseline system is case 5 and we pose the same question: Is it better to improve the components or to improve the FDI system? Case 2 has better components with the same FDI performance, while case 4 uses better FDI with the same components. The superimposed outlines of the three PMFs in Figure 4.33 show that improving the components is the better option. The case 2 PMF is an order of magnitude below the case 5 PMF over its entire range, while the case 4 PMF offers only a small improvement over case 5 in the roll-off region.

When performance evaluation is used to design RM systems, the comparison of different systems may be quite different. The models in the six cases considered here all have the same performance measures, so the PMFs had approximately the same ranges of performance values when the same look-ahead tolerance was used. It is more likely that this would not be the case in a real application. For example, improving the components may change not only the probabilistic structure of the model, but also the performance structure if more reliable components also have better accuracy. The comparison of systems involving such component variations will be clear using performance PMFs.

Performance PMFs emphasize several design goals. Primarily, it is best to have most of the probability in the PMF concentrated in the lower range of the performance scale. Designing systems with low, narrow plateaus and sharp roll-off regions is preferable to high, wide plateaus and slow roll-offs. It is also beneficial to use a number of different performance evaluation structures to evaluate a system. Performance evaluation results are as much a function of the performance structure of a

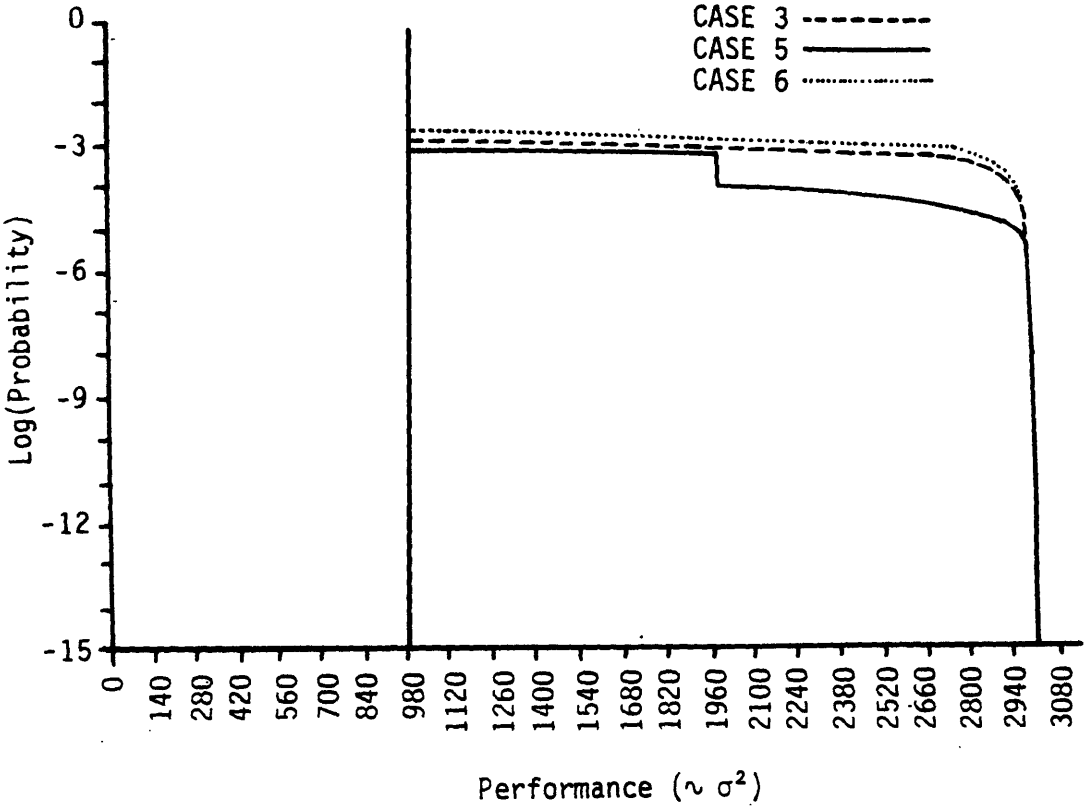


Figure 4.32. PMF Comparison for Cases 3, 5, and 6

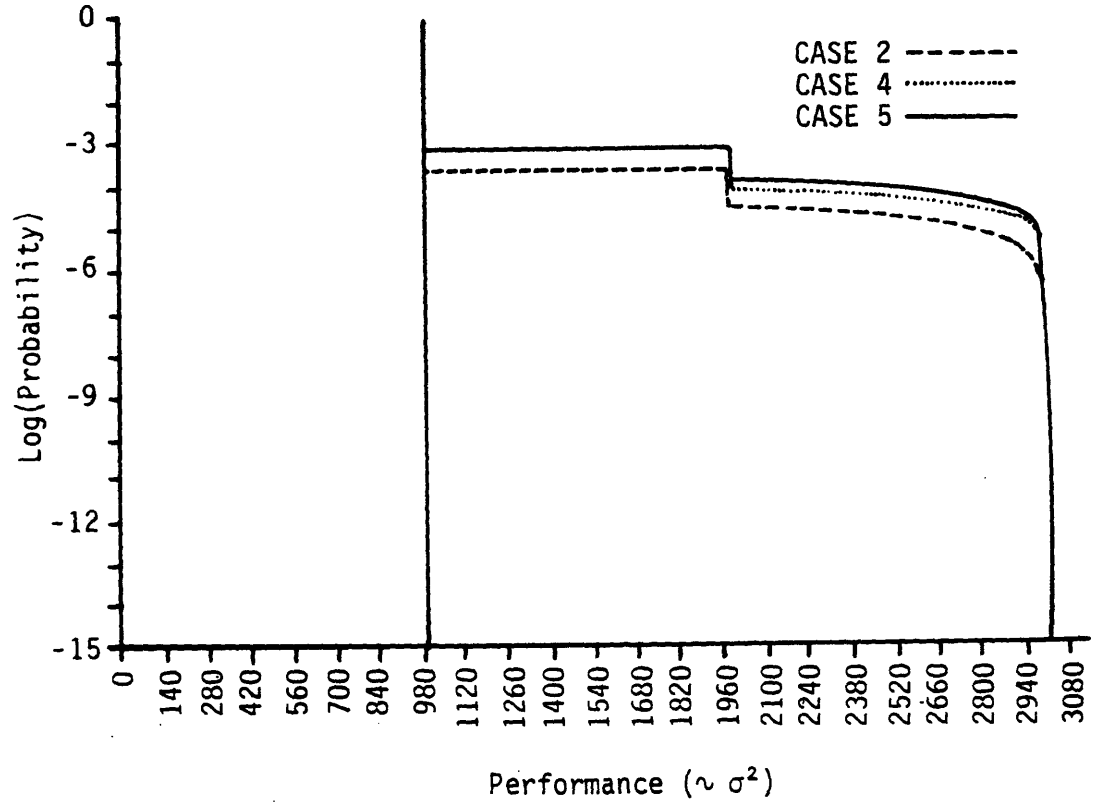


Figure 4.33. PMF Comparison for Cases 2, 4, and 5

model as they are a function of the probabilistic structure. There is certainly more than one performance measure which can be derived for each model state for most systems, and therefore it will be wise to compare PMFs for as many different sets of performance criteria as are deemed relevant to the design. During all comparisons, however, it is a good idea to keep the tolerance approximately the same relative to $v_1(k_m)$ to insure that the PMFs change appearance due to different models and not different tolerances.

CHAPTER 5

CONCLUSION

5.1 Summary

The Markov models of fault-tolerant systems which use redundancy management are inherently transient models with trivial steady-state behavior. New methods are needed to analyze such systems because the large body of theory which exists for analyzing processes with non-trivial steady-state behavior does not apply in general to these transient systems.

This thesis has developed and demonstrated two new methods for evaluating the performance of fault-tolerant systems using transient Markov models. Beginning with the concept of operational state histories and OSH ensembles, it was first demonstrated that ensemble growth in time is at worst linear and not exponential if an integral performance measure is adopted. The newly conceived v-transform was introduced to represent and manipulate OSH ensembles and to allow straightforward calculation of performance PMFs. Second, the theory of Markov processes with rewards was adapted to calculate the transient expected performance profile and the total expected performance vector of a system. Finally, the interaction of the two methods supplied a basis for making the conservative, predictable look-ahead approximation which simplifies the calculation of performance PMFs.

To demonstrate the usefulness and feasibility of the performance methods developed, computer programs were written which manipulate matrices of v-transforms and calculate total expected performance profiles. Using these programs, several example models were analyzed. A 7-state Markov

model demonstrated the basic theoretical concepts and revealed the form of typical performance PMFs and total expected performance profiles. Next, the analysis of a 50-state model demonstrated that performance evaluation can permit reasonable approximations to be made during the modeling process. A 10-state approximation to the 50-state model produced essentially the same results from the performance evaluation standpoint. The primary use of the performance evaluation techniques developed here is in RM system design. Hence, several performance PMFs and profiles were calculated for an 8 state model for variations in its key parameters (MTTFs and P_{FA}). Parameter variation showed that the performance evaluation methods are good design tools for comparing different RM designs. Given a certain RM design, these methods can also indicate to the engineer which potential improvements to the system will bring about the greatest marginal increase in system performance.

Though expected performance statistics and profiles are easier to compute than performance PMFs, to use them as the sole indicator of system performance is dangerous. First, because the failure times of components are generally exponentially distributed, the variance of the expected performance will be equal to or much larger than the expected performance itself. Second, the total expected performance of the system does not reveal the fraction of total expected performance represented by OSHs which occupy operational states at mission time and the expected performance of the SLOSHs. The decomposition of the total expected performance is important. Finally, the performance PMF is a rich source of design information. It is conceivable that changes in an RM system design could leave the total expected performance value unchanged while drastically changing the shape of the performance PMF and the behavior of the system.

In such a case, the total expected performance would be particularly misleading.

5.2 Recommendations

V-transform analysis is very flexible, especially when the manipulations are done symbolically in a LISP environment. Many modeling variations such as time-varying performance measures or time-varying component and FDI behavior can be added with no computational penalty. There are also many other possibilities for approximations similar to the look-ahead approximation which simplify computation without affecting the results drastically. For example, an approximation which redistributes culled OSHs, combining them with other OSHs with similar cumulative performances rather than discarding them, may prove to be better than the look-ahead approximation.

As stated in Chapter One, showing how to construct transient Markov models was not a major goal of this thesis. However, it became quite clear during the course of this research that the greatest challenge in evaluating the performance of a system may be the specification of an accurate model for that system. Once the model is specified, applying the techniques developed in this thesis should be straightforward. The importance of accurate modeling cannot be overemphasized. The results of performance evaluation can only be as accurate as the model itself.

In the interest of constructing example models, several assumptions were made which will not be true in general. For example, it was assumed that all components in each system were identical, that the subsystems for the 50-state model were independent with respect to the performance

measures and state transition probabilities, and that components have independent accuracies and MTTFs. Furthermore, closed-loop plant dynamics and reconfiguration algorithms were not considered in any of the performance measures. The next logical step in the modeling area will be to locate systems which are either operational or in advanced design stages and to try to model them realistically for the purposes of performance evaluation.

Great improvements can be made in the automated manipulation of v-transforms. The SCHEME system provided in Appendix B served only to demonstrate the feasibility of the v-transform and performance PMF concepts. Though the linear growth of OSH ensembles in time is comforting, computation time is proportional to k^2. This causes long execution times for complex systems. Though the look-ahead approximation helps, computational improvements are clearly necessary. An efficient v-transform manipulation package should be written for use on a larger computer with virtual memory, preferably a dedicated LISP processor. Though results cannot be guaranteed for ever-increasing model dimensions or mission times, this single advancement will probably overcome the computational complexity of problems which will be encountered while analyzing current systems and those of the near future.

Thinking ahead to the long-term evolution of performance evaluation methods, v-transform manipulation is ideally suited to the parallel, multiprocessor computers which are currently envisioned. Since the entire matrix of v-transforms is known a priori, the v-transforms for different parts of an OSHE and for different times k could be calculated simultaneously and recombined to generate the performance PMF. Such a system would permit larger models to be analyzed for longer mission times

without significantly increasing the computation time, and without the need for storing all results in memory simultaneously. These benefits make parallel processing of v-transform matrices a quite attractive possibility.

APPENDIX A

MODELING CALCULATIONS

APPENDIX A

Modeling calculations for the 8-state "double star" actuator and sensor subsystem models described in Section 4.3.

1. Mission time:

   Mission Duration: 2 hours
   FDI Test Period: 1 second
   k = 2 x 60 x 60 = 7200 FDI tests per mission

2. Failure Probabilities:
   Single component failure times are exponentially distributed:

$$P(F_1) = 1 - e^{-\frac{t}{MTTF}}$$

   For t = ΔT, the FDI test period:

$$e^{-\frac{\Delta T}{MTTF}} \simeq 1 - \frac{\Delta T}{MTTF}$$

   Hence, the single component failure probability during ΔT is

$$P(F_1) \simeq 1 - (1 - \frac{\Delta T}{MTTF})$$

$$= \frac{\Delta T}{MTTF}$$

   Failure probability for 1 of n components during ΔT, assuming all components are identical:

$$P(F_n) = 1 - P(\text{each component does } \underline{\text{not}} \text{ fail})$$

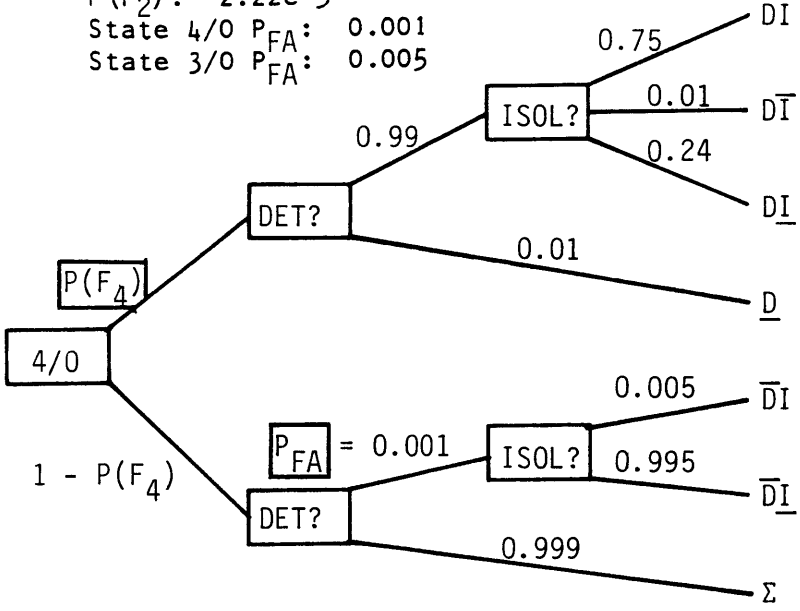
$$= 1 - (1 - P(F_1))^n$$

$$= 1 - (1 - \frac{\Delta T}{MTTF})^n$$

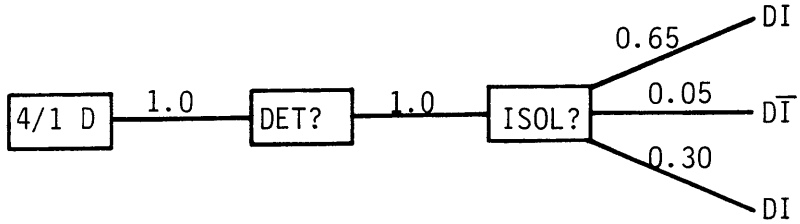
$$\simeq 1 - (1 - n\frac{\Delta T}{MTTF}), \quad \text{if } \frac{\Delta T}{MTTF} << 1$$

$$= n\frac{\Delta T}{MTTF}$$

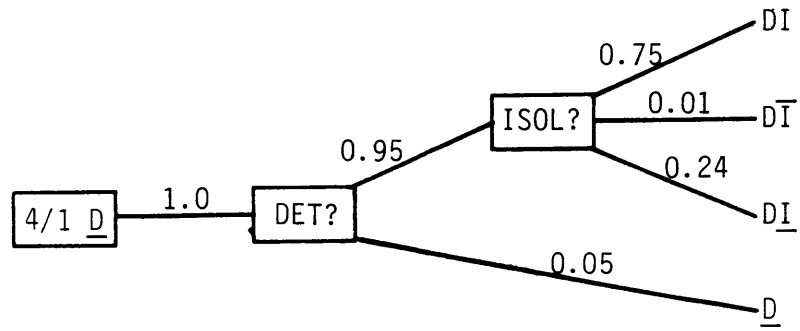
3. Example event trees for each state of the actuator subsystem:
   4-component system / 2 functional components required for operation.
   MTTF: 25 hours (9.0e+4 seconds)
   ΔT: 1 second
   $P(F_4)$: 4.44e-5
   $P(F_3)$: 3.33e-5
   $P(F_2)$: 2.22e-5
   State 4/0 P_{FA}: 0.001
   State 3/0 P_{FA}: 0.005



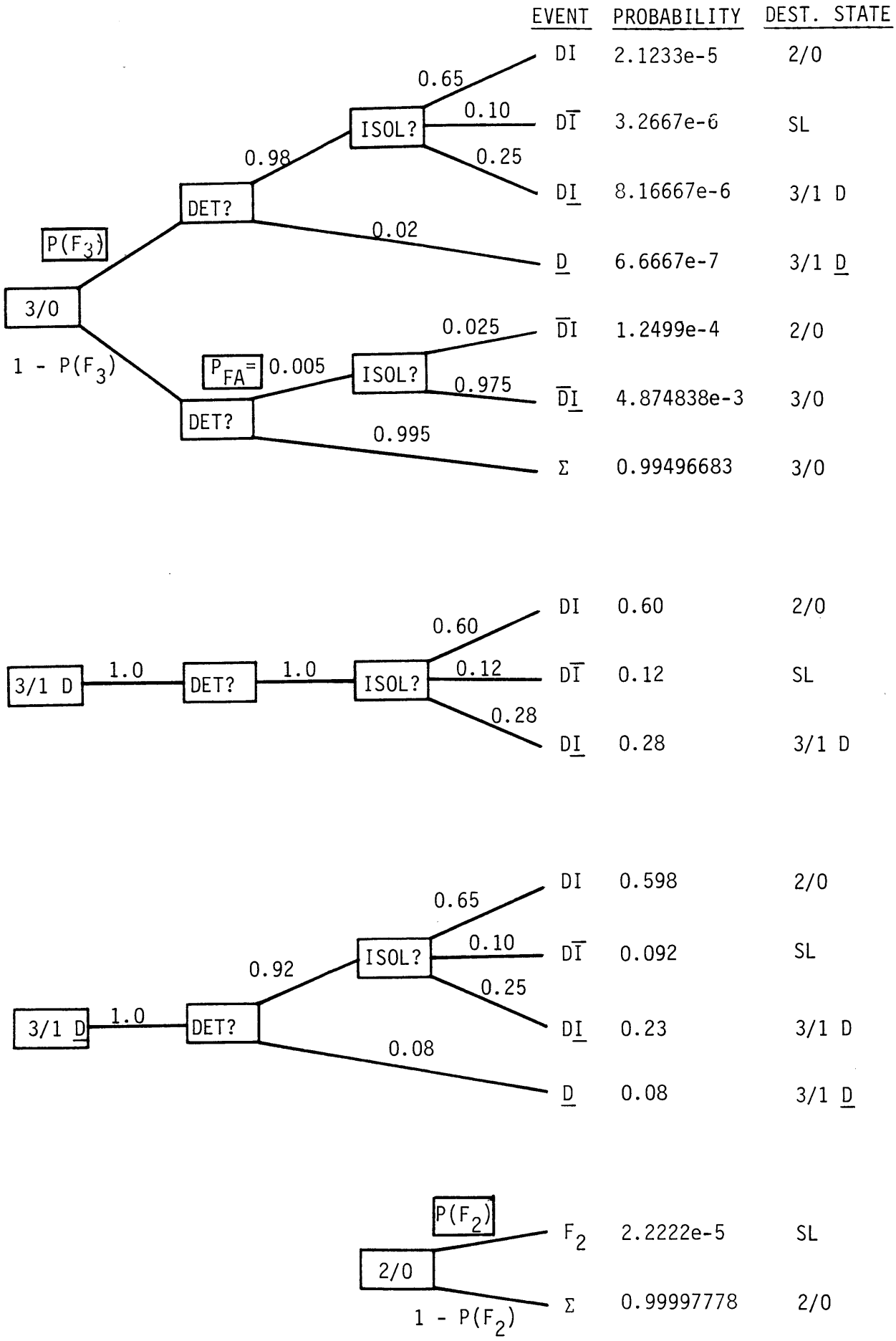
EVENT	PROBABILITY	DEST. STATE
DI	3.3e-5	3/0
$D\bar{I}$	4.4e-7	3/1 D
DI	1.056e-5	4/1 D
D	4.4e-7	4/1 D
$\bar{D}I$	4.9997e-6	3/0
$\bar{D}\bar{I}$	9.94956e-4	4/0
Σ	0.99895560	4/0



DI	0.65	3/0
$D\bar{I}$	0.05	3/1 D
DI	0.30	4/1 D



DI	0.7125	3/0
$D\bar{I}$	0.0095	3/1 D
DI	0.228	4/1 D
D	0.05	4/1 D


EVENT
PROBABILITY
DEST. STATE
3/0
P(F3)
1 - P(F3)
DET?
0.98
0.02
ISOL?
0.65
0.10
0.25
PFA=
0.005
0.995
0.025
0.975
DI
2.1233e-5
2/0
DI
3.2667e-6
SL
DI
8.16667e-6
3/1 D
D
6.6667e-7
3/1 D
DI
1.2499e-4
2/0
DI
4.874838e-3
3/0
Σ
0.99496683
3/0
3/1 D
1.0
DET?
1.0
ISOL?
0.60
0.12
0.28
DI
0.60
2/0
DI
0.12
SL
DI
0.28
3/1 D
3/1 D
1.0
DET?
0.92
0.08
ISOL?
0.65
0.10
0.25
DI
0.598
2/0
DI
0.092
SL
DI
0.23
3/1 D
D
0.08
3/1 D
P(F2)
2/0
1 - P(F2)
F2
2.2222e-5
SL
Σ
0.99997778
2/0

The resulting single-step transition matrix for the actuator subsystem:

0.999950558						
4.4444e-7	0.50					
1.056e-5	0.228	0.30				
3.79998e-5	0.7125	0.65	0.99984167			
4.4e-7	0.0095	0.05	6.66667e-7	0.08		
0			8.16667e-6	0.23	0.28	
			1.46229e-4	0.598	0.60	0.999977778
			3.26667e-6	0.092	0.12	2.2222e-5

4. State transition matrices for the sensor subsystems:
The event tress and calculations are identical to the actuators subsystems except for changes in $P(F_4)$, $P(F_3)$, $P(F_2)$, and the P_{FA}s for states 4/0 and 3/0. These probabilities are highlighted with boxes in the event tree diagrams above. The general form of the state transition matrix represented by the event trees above is shown below. Each entry in the matrix represents the event or events which causes the transition to occur.

ORIGIN STATE

DESTINATION STATE	4/0	4/1 $\underline{D}$	4/1 D	3/0	3/1 $\underline{D}$	3/1 D	2/0	SL
4/0	$\overline{D}\underline{I} + \Sigma$							
4/1 $\underline{D}$	$\underline{D}$	$\underline{D}$						
4/1 D	$D\underline{I}$	$D\underline{I}$	$D\underline{I}$			0		
3/0	$DI + \overline{D}I$	DI	DI	$\overline{D}\underline{I} + \Sigma$				
3/1 $\underline{D}$	$D\overline{I}$	$D\overline{I}$	$D\overline{I}$	$\underline{D}$	$\underline{D}$			
3/1 D		0		$D\underline{I}$	$D\underline{I}$	$D\underline{I}$		
2/0				$DI + \overline{D}I$	DI	DI	$1 - P(F_2)$	
SL				$D\overline{I}$	$D\overline{I}$	$D\overline{I}$	$P(F_2)$	1.0

CASE 1 (Used with the actuators to form the 50-state model in Section 4.4.2, and for the parameter variation analysis of Section 4.4.3.

MTTF: 100 hours
$P(F_4)$: 1.11e-5
$P(F_3)$: 8.33e-6
$P(F_2)$: 5.55e-6
State 4/0 P_{FA}: 0.001
State 3/0 P_{FA}: 0.005

Single-step state transition matrix:

$$\begin{bmatrix}
0.999983889 & & & & & & & \\
1.1111e-7 & 0.05 & & & & & & \\
2.64e-6 & 0.228 & 0.30 & & & & & \\
1.32499e-5 & 0.7125 & 0.65 & 0.9998413 & & & & \\
1.1e-7 & 0.0095 & 0.05 & 1.66667e-7 & 0.08 & & & \\
0 & & & 2.04167e-6 & 0.23 & 0.28 & & \\
 & & & 1.30307e-4 & 0.598 & 0.60 & 0.999994444 & \\
 & & & 8.16667e-7 & 0.092 & 0.12 & 5.55556e-6 & 1.0
\end{bmatrix}$$

CASE 2

MTTF: 100 hours
$P(F_4)$: 1.11e-5
$P(F_3)$: 8.33e-6
$P(F_2)$: 5.55e-6
State 4/0 P_{FA}: 0.005
State 3/0 P_{FA}: 0.01

Single-step state transition matrix:

$$\begin{bmatrix}
0.99996390 & & & & & & & \\
1.1111e-7 & 0.05 & & & & & & \\
2.64e-6 & 0.228 & 0.30 & & & & & \\
3.32497e-5 & 0.7125 & 0.65 & 0.999741669 & & & & \\
1.1e-7 & 0.0095 & 0.05 & 1.66667e-7 & 0.08 & & & \\
0 & & & 2.04167e-6 & 0.23 & 0.28 & & \\
 & & & 2.55306e-4 & 0.598 & 0.60 & 0.999994444 & \\
 & & & 8.16667e-7 & 0.092 & 0.12 & 5.55556e-6 & 1.0
\end{bmatrix}$$

CASE 3

MTTF: 100 hours
$P(F_4)$: 1.11e-5
$P(F_3)$: 8.33e-6
$P(F_2)$: 5.55e-6
State 4/0 P_{FA}: 0.05
State 3/0 P_{FA}: 0.1

Single-step state transition matrix:

$$\begin{bmatrix}
0.999738905 & & & & & & & \\
1.1111e-7 & 0.05 & & & & & & \\
2.64e-6 & 0.228 & 0.30 & & & & & \\
2.58247e-4 & 0.7125 & 0.65 & 0.997491630 & & & & \\
1.1e-7 & 0.0095 & 0.05 & 1.66667e-7 & 0.08 & & & \\
0 & & & 2.04167e-6 & 0.23 & 0.28 & & \\
 & & & 2.50528e-3 & 0.598 & 0.60 & 0.999994444 & \\
 & & & 8.16667e-7 & 0.092 & 0.12 & 5.55556e-6 & 1.0
\end{bmatrix}$$

CASE 4

MTTF: 10 hours
$P(F_4)$: 1.11e-4
$P(F_3)$: 8.33e-5
$P(F_2)$: 5.55e-5
State 4/0 P_{FA}: 0.001
State 3/0 P_{FA}: 0.005

Single-step state transition matrix:

$$\begin{bmatrix}
0.9998838894 & & & & & & & \\
1.1111e-6 & 0.05 & & & & & & \\
2.64e-5 & 0.228 & 0.30 & & & & & \\
8.74994e-5 & 0.7125 & 0.65 & 0.999791677 & & & & \\
1.1e-6 & 0.0095 & 0.05 & 1.66667e-6 & 0.08 & & & \\
0 & & & 2.04167e-5 & 0.23 & 0.28 & & \\
 & & & 1.78072e-4 & 0.598 & 0.60 & 0.999944444 & \\
 & & & 8.16667e-6 & 0.092 & 0.12 & 5.55556e-5 & 1.0
\end{bmatrix}$$

CASE 5

MTTF: 10 hours
$P(F_4)$: 1.11e-4
$P(F_3)$: 8.33e-5
$P(F_2)$: 5.55e-5
State 4/0 P_{FA}: 0.005
State 3/0 P_{FA}: 0.01

Single-step state transition matrix:

$$\begin{bmatrix} 0.999863891 & & & & & & & \\ 1.1111e\text{-}6 & 0.05 & & & & & & \\ 2.64e\text{-}5 & 0.228 & 0.30 & & & 0 & & \\ 1.07497e\text{-}4 & 0.7125 & 0.65 & 0.999666688 & & & & \\ 1.1e\text{-}6 & 0.0095 & 0.05 & 1.66667e\text{-}6 & 0.08 & & & \\ & 0 & & 2.04167e\text{-}5 & 0.23 & 0.28 & & \\ & & & 3.03062e\text{-}4 & 0.598 & 0.60 & 0.999944444 & \\ & & & 8.16667e\text{-}6 & 0.092 & 0.12 & 5.55556e\text{-}5 & 1.0 \end{bmatrix}$$

CASE 6

MTTF: 10 hours
$P(F_4)$: 1.11e-4
$P(F_3)$: 8.33e-5
$P(F_2)$: 5.55e-5
State 4/0 P_{FA}: 0.05
State 3/0 P_{FA}: 0.1

Single-step state transition matrix:

$$\begin{bmatrix} 0.99963891 & & & & & & & \\ 1.1111e\text{-}6 & 0.05 & & & & & & \\ 2.64e\text{-}5 & 0.228 & 0.30 & & & 0 & & \\ 3.32472e\text{-}4 & 0.7125 & 0.65 & 0.99741688 & & & & \\ 1.1e\text{-}6 & 0.0095 & 0.05 & 1.66667e\text{-}6 & 0.08 & & & \\ & 0 & & 2.04167e\text{-}5 & 0.23 & 0.28 & & \\ & & & 2.55287e\text{-}3 & 0.598 & 0.60 & 0.999944444 & \\ & & & 8.16667e\text{-}6 & 0.092 & 0.12 & 5.55556e\text{-}5 & 1.0 \end{bmatrix}$$

5. Performance Value Calculations for Both Subsystems:

Performance values for the sensor subsystem are proportional to the variance of the measurement noise for each state. A failure increases the measurement noise of a single sensor by $9\sigma^2$. Below, the variances of the measurement noise for each model state are calculated. These values are used both in Section 4.4.2 and in Section 4.4.3.

Operating Sensor Measurement Noise: $N(0, \sigma^2)$
Failed Sensor Measurement Noise: $N(0, 10\sigma^2)$

State 4/0: $\frac{4\sigma^2}{16} = \frac{\sigma^2}{4}$

States 4/1 D and 4/1 D: $\frac{10\sigma^2 + \sigma^2 + \sigma^2 + \sigma^2}{16} = \frac{13}{16}\sigma^2$

State 3/0: $\frac{3\sigma^2}{9} = \frac{1}{3}\sigma^2$

States 3/1 D and 3/1 D: $\frac{10\sigma^2 + \sigma^2 + \sigma^2}{9} = \frac{4}{3}\sigma^2$

State 2/0: $\frac{2\sigma^2}{4} = \frac{\sigma^2}{2}$

Dividing the above variances by σ^2 and fitting the resulting numbers to an integral scale yields the following normalized performance values:

State 4/0: 12
States 4/1 D and 4/1 D: 39
State 3/0: 16
States 3/1 D and 3/1 D: 64
State 2/0: 24

Performance values for the actuator subsystem used in Section 4.4.2 were fixed relative to the values for the sensor subsystem according to the following rules:

1. Actuators are 10 times more critical to mission completion than the sensors, so the actuator state 4/0 value is 10 times the sensor state 4/0 value.

2. A detected, isolated failure increases the performance value by a factor of $\sqrt{2}$.

3. A missed detection increases the performance value by a factor of 10.

4. A missed isolation increases the performance value by a factor of 5.

The following performance values can then be calculated:

Sensor state 4/0 value = 12
Actuator state 4/0 value = 120
Actuator state 4/0 $\underline{D}$ = 1200
Actuator state 4/0 $\bar{D}$ = 600
Actuator state 3/0 = 170
Actuator state 3/0 $\underline{D}$ = 1700
Actuator state 3/0 $\bar{D}$ = 850
Actuator state 2/0 = 240

Because the OSHE can potentially grow in time at the rate of the largest performance value, all values were scaled down by a factor of 10 to yield the values which were used in Sections 4.4.2 and 4.4.3:

State	Sensors	Actuators
x_1	1	12
x_2	4	120
x_3	4	60
x_4	2	17
x_5	7	170
x_6	7	85
x_7	3	24

APPENDIX B

COMPUTER PROGRAM LISTINGS

```
001   ;;  This is the listing of the file "A
      ;PPROX.SCM."  This file has all of the
002   ;;  procedures necessary for manipulating matrices of v-transforms in a
003   ;;  number of ways.  Provision is also included to cull OSHs using
004   ;;  the look-ahead approximation.  The file "rule-list5.scm" can
005   ;;  be used with this file.  Any other file which defines a matrix must also
006   ;;  include bindings for "vkm" and "tolerance."
007       (declare (usual-integrations))
008
009
010   ;; Top level (level 1) procedures:  Operations on matrices
011
012   ;; Gen-pmf returns a performance transform given a matrix of v-transforms, the
013   ;; number of time steps desired, the beginning state (which must be one of the
014   ;; column designators of the matrix), the total expected performance vector at
015   ;; mission time, vkm, and a tolerance, tol.  Vkm should be an alist with column
016   ;; designators as the keys.
017
018   (define (gen-pmf matrix steps begin-state vkm tol)
019     (let loop ((result (make-init begin-state)) (count steps))
020       (cond ((zero? count)
021              (transform (add-elements result)))
022             (else
023              (print count)
024              (set! result (merge-column (multiply-col-mtx result matrix vkm tol)))
025              (loop result (-1+ count))))))
026
027   ;; Gen-pmf-with-stats also computes the performance transform, but in addition
028   ;; returns lists of statistics for each step of propagation.  The statistics
029   ;; returned are the expected performance of the PMF, the expected performance
030   ;; arriving at system-loss, the expected performance of culled terms, the total
031   ;; expected performance, the total culled probability, and the unreliability.
032
033
034   (define (gen-pmf-with-stats matrix steps begin-state vkm tolerance)
035     (let loop ((result (make-init begin-state)) (count steps)
036                (pmf-jbar '()) (syst-loss-jbar '()) (culled-jbar '())
037                (total-jbar '()) (culled-prob '()) (unreliability '()))
038       (cond ((zero? count)
039              (append (list pmf-jbar) (list syst-loss-jbar) (list culled-jbar)
040                      (list total-jbar) (list culled-prob) (list unreliability)
041                      (list (transform (add-elements result)))))
042             (else
043              (print count)
044              (set! result (merge-column (multiply-col-mtx result matrix
045                                                           vkm tolerance)))
046              (let ((t (car (transform (assoc 'system-loss (elements result)))))
047                    (c (car (transform (assoc 'culled (elements result)))))
048                    (pjbr (pmf-j-bar result)))
049                (set! pmf-jbar (cons pjbr pmf-jbar))
050                (cond ((null? t)
051                       (set! t '(0 0))))
052                (cond ((null? c)
053                       (set! c '(0 0))))
054                (set! syst-loss-jbar (cons (expon t) syst-loss-jbar))
055                (set! culled-jbar (cons (expon c) culled-jbar))
056                (set! total-jbar (cons (+ pjbr (expon t) (expon c))
057                                       total-jbar))
058                (set! culled-prob (cons (coef c) culled-prob))
059                (set! unreliability (cons (coef t) unreliability)))
060              (loop result (-1+ count) pmf-jbar syst-loss-jbar culled-jbar
061                    total-jbar culled-prob unreliability)))))
062
063
064   ;; Gpws is another version of the procedure above.  It functions identically.
065
066   (define (gpws matrix steps begin-state vkm tolerance)
067     (let loop ((result (make-init begin-state)) (count steps)
068                (pmf-jbar '()) (syst-loss-jbar '()) (culled-jbar '())
069                (total-jbar '()) (culled-prob '()) (unreliability '()))
070       (cond ((zero? count)
071              (append (list pmf-jbar) (list syst-loss-jbar) (list culled-jbar)
072                      (list total-jbar) (list culled-prob) (list unreliability)
073                      (list (transform (add-elements result)))))
074             (else
075              (print count)
076              (set! result (merge-column (multiply-col-mtx result matrix
```

```
                                                              vkm tolerance)))
            (let ((t (car (transform (assoc 'system-loss (elements result)))))
                  (c (car (transform (assoc 'culled (elements result)))))
                  (pjbr (pmf-j-bar result)))
              (cond ((null? t)
                     (set! t '(0 0))))
              (cond ((null? c)
                     (set! c '(0 0))))
              (loop result (-1+ count) (cons pjbr pmf-jbar)
                    (cons (expon t) syst-loss-jbar) (cons (expon c) culled-jbar)
                    (cons (- pjbr (expon t) (expon c)) total-jbar)
                    (cons (coef c) culled-prob) (cons (coef t) unreliability)))))))


;; Power computes the performance transform using a combination of gen-pmf and
;; exponentiate.  The spec-list takes the place of "steps."  The spec-list
;; should be a list of numbers indicating the progressive exponentiations
;; desired of the input matrix, mtx.  On the last specification, the procedure
;; gen-pmf is used in order to save space.  The product of the number in the
;; spec-list should be the number of steps for which the performance transform
;; is desired.

(define (power matrix spec-list begin-state vkm tolerance)
  (let loop ((specs spec-list) (mtx matrix))
    (cond ((zero? (- (length specs) 1))
           (gen-pmf mtx (car specs) begin-state
                            vkm tolerance))
          (else
           (loop (cdr specs)
                 (exponentiate mtx (car specs) vkm tolerance))))))

;; Exponentiate raises the specified matrix to the specified power.

(define (exponentiate matrix power vkm tolerance)
  (let loop ((result matrix) (count (-1+ power)))
    (cond ((zero? count)
           result)
          (else
           (print count)
           (set! result (multiply-matrices result matrix vkm tolerance))
           (loop result (-1+ count))))))

;;  Multiply-matrices returns the matrix which is the product of matrix1
;;  and matrix2.

(define (multiply-matrices matrix1 matrix2 vkm tolerance)
  (let ((result '() ))
    (let loop ((matrix matrix1))
      (cond ((null? matrix)
             result)
            (else
             (set! result (cons (merge-column (multiply-col-mtx (car matrix)
                                                                matrix2
                                                                vkm
                                                                tolerance))
                                result))
             (loop (cdr matrix)))))))


(define (get-column column-number matrix)
  (assoc column-number matrix))



;; Level 2: Operations on columns

(define (multiply-col-mtx column matrix vkm tol)   ;; Multiplies a column by a
  (let ((result '() ))                             ;; matrix and returns the elements
    (let loop1 ((els (elements column)))           ;; of the resulting column.
      (cond ((null? els)
             (make-column (column-number column) (cull result vkm tol)))
            ((culled? (car els))
             (set! result (cons (car els) result))
             (loop1 (cdr els)))
            (else
             (let loop2 ((product (multiply-el-col (car els)
```

```
                                                    (get-column (row-number
                                                                 (car els))
                                                                matrix))))
                (if (null? product)
                    (loop1 (cdr els))
                    (let ((element (assoc (row-number (car product))
                                          result)))
                      (cond ((null? element)
                             (set! result (cons (car product) result))
                             (loop2 (cdr product)))
                            (else
                             (set-cdr! element (list (append (transform element)
                                                             (transform
                                                              (car product)))))
                             (loop2 (cdr product))))))))))

(define (merge-column column)                ;; Returns column whose elements have
  (let ((result '()))                        ;; had either the merge-element, the
    (let loop ((els (elements column)))      ;; merge-culled, or the merge-syst-
      (cond ((null? els)                     ;; loss procedures applied to them.
             (make-column (column-number column) result))
            ((system-loss? (car els))
             (set! result (cons (merge-syst-loss (car els))
                                result))
             (loop (cdr els)))
            ((culled? (car els))
             (set! result (cons (merge-culled (car els))
                                result))
             (loop (cdr els)))
            (else
             (set! result (cons (merge-element (car els))
                                result))
             (loop (cdr els)))))))

(define (add-elements column)
  (let ((result '()))
    (let loop ((els (elements column)))
    (cond ((null? els)
           (merge-element (make-element (column-number column) result)))
          ((culled? (car els))
           (loop (cdr els)))
          ((system-loss? (car els))
           (loop (cdr els)))
          (else
           (set! result (append (transform (car els)) result))
           (loop (cdr els)))))))

(define (make-init begin-state)
  (list begin-state (list (list begin-state (list (list 0 1))))))

(define (pmf-j-bar column)
  (let loop ((j-bar 0) (els (elements column)))
    (cond ((null? els)
           j-bar)
          ((system-loss? (car els))
           (loop j-bar (cdr els)))
          ((culled? (car els))
           (loop j-bar (cdr els)))
          (else
           (set! j-bar (+ j-bar (ddvat1 (transform (car els)))))
           (loop j-bar (cdr els))))))

(define (count-terms column)
  (let ((count 0))
    (let loop ((els (elements column)))
      (cond ((null? els)
             count)
            ((system-loss? (car els))
             (loop (cdr els)))
            ((culled? (car els))
             (loop (cdr els)))
            (else
             (set! count (+ count (length (transform (car els)))))
             (loop (cdr els)))))))

(define (column-number column)
```

```
  (car column))

(define (elements column)
  (cadr column))

(define (make-column column-number elements)
  (list column-number elements))



;; Level 3:  Operations on elements

(define (multiply-elements element1 element2)
  (let ((result '() ))
    (let loop1 ((tf1 (transform element1)) (tf2 (transform element2)))
      (cond ((null? tf1)
             (cond ((eq? (row-number element1) 'systen-loss)
                    (make-element 'system-loss result))
                   ((eq? (row-number element2) 'systen-loss)
                    (make-element 'system-loss (reduce-transform result)))
                   (else
                    (make-element (row-number element2) result))))
            (else                                      ;; Multiplies two elements
             (let loop2 ((t2 tf2))                     ;; and returns the resulting
               (cond ((null? t2)                       ;; element, which takes the
                      (loop1 (cdr tf1) tf2))           ;; row number of element2.
                     (else
                      (set! result (cons (multiply-terms (car tf1) (car t2))
                                         result))
                      (loop2 (cdr t2)))))))))))

(define (multiply-el-col element column)       ;; Multiplies the transform of
  (let ((result '() ))                         ;; one element by the transforms
    (let loop ((els (elements column)))        ;; of the elements of the column
      (cond ((null? els)                       ;; and returns the list of
             result)                           ;; resulting elements.
            ((culled? (car els))
             (loop (cdr els)))
            (else
             (set! result (cons (multiply-elements element (car els))
                                result))
             (loop (cdr els)))))))

(define (merge-element element)                ;; Returns the element which
  (let ((result '() ))                         ;; results when all terms in the
    (let loop ((tf (transform element)))       ;; argument which have the same
      (if (null? tf)                           ;; exponent are added.
          (make-element (row-number element) result)
          (let ((term (assq (expon (car tf)) result)))
            (cond ((null? term)
                   (set! result (cons (car tf) result))
                   (loop (cdr tf)))
                  (else
                   (set-cdr! term (list (+ (coef term)
                                           (coef (car tf)))))
                   (loop (cdr tf)))))))))

(define (merge-syst-loss element)              ;; Creates a syst-loss datum
  (let ((sum-of-probs 0) (costbar 0))          ;; of the syst-loss trap prob
    (let loop ((tf (transform element)))       ;; and cumulative expected cost
      (cond ((null? tf)                        ;; of trajs ending at syst-loss.
             (make-element 'system-loss (list (make-term
                                               costbar
                                               sum-of-probs))))
            (else
             (set! sum-of-probs (+ (coef (car tf)) sum-of-probs))
             (set! costbar (+ (expon (car tf)) costbar))
             (loop (cur tf)))))))

(define (merge-culled element)
  (let ((sum-of-probs 0) (costbar 0))
    (let loop ((tf (transform element)))
      (cond ((null? tf)
             (make-element 'culled (list (make-term costbar sum-of-probs))))
            (else
```

```
305               (set! sum-of-probs (+ (coef (car tf)) sum-of-probs))
306               (set! costbar (+ (expon (car tf)) costbar))
307               (loop (cdr tf)))))))
308
309   (define (cull el-list vkm tolerance)
310     (let ((result '()))
311       (let loop ((els el-list))
312         (cond ((null? els)
313                result)
314               (else
315                (let ((el (car els)))
316                  (cond ((system-loss? el)
317                         (set! result (cons el result))
318                         (loop (cdr els)))
319                        ((culled? el)
320                         (let ((cull-el (assoc 'culled result)))
321                           (cond ((null? cull-el)
322                                  (set! result (cons el result))
323                                  (loop (cdr els)))
324                                 (else
325                                  (set-cdr! cull-el
326                                            (list (append (transform el)
327                                                          (transform cull-el))))
328                                  (loop (cdr els))))))
329                        (else
330                         (let ((culled-el (cull-transform (transform el)
331                                                          (cadr (assoc (row-number
332                                                                        el)
333                                                                       vkm))
334                                                          tolerance))
335                               (cull-el (assoc 'culled result)))
336                           (set! result (cons (make-element (row-number el)
337                                                            (transform
338                                                             (cadr culled-el)))
339                                              result))
340                           (cond ((null? cull-el)
341                                  (set! result (cons (car culled-el) result))
342                                  (loop (cdr els)))
343                                 (else
344                                  (set-cdr! cull-el (list (append
345                                                           (transform cull-el)
346                                                           (transform
347                                                            (car culled-el)))))
348                                  (loop (cdr els)))))))))))
349
350   (define (system-loss? element)
351     (eq? (row-number element) 'system-loss))
352
353   (define (culled? element)
354     (eq? (row-number element) 'culled))
355
356   (define (row-number element)
357     (car element))
358
359   (define (transform element)
360     (cadr element))
361
362   (define (make-element row-number transform)
363     (list row-number transform))
364
365
366
367
368   ;; Level 4:  Operations on transforms
369
370   (define (cull-transform transform value-remaining tolerance)
371     (let ((culls '()) (oks '()))
372       (let loop ((terms transform))
373         (cond ((null? terms)
374                (list (list 'culled (reduce-transform culls)) (list 'oks oks)))
375               ((low-perf? (car terms) value-remaining tolerance)
376                (set! culls (cons (car terms) culls))
377                (loop (cdr terms)))
378               (else
379                (set! oks (cons (car terms) oks))
380                (loop (cdr terms)))))))
```

```
381
382  (define (reduce-transform transform)
383    (list (make-term (ddvat1 transform) (add-coef transform))))
384
385  (define (add-coef transform)            ;; Returns the sum of the coefficients
386    (if (null? transform)                 ;; of the specified transform.
387        0
388        (+ (coef (car transform)) (add-coef (cdr transform)))))
389
390  (define (ddvat1 transform)              ;; Evaluates the derivative of the
391    (if (null? transform)                 ;; specified cost transform at v=1.
392        0
393        (+ (* (coef (car transform))
394              (expon (car transform)))
395           (ddvat1 (cdr transform)))))
396
397  (define (d2dv2at1 transform)
398    (if (null? transform)
399        0
400        (+ (* (coef (car transform))
401              (expon (car transform))
402              (expon (car transform)))
403           (d2dv2at1 (cdr transform)))))
404
405  (define (variance transform)
406    (let ((ej (ddvat1 transform))
407          (ej2 (d2dv2at1 transform)))
408      (- ej2 (* ej ej))))
409
410
411  ;; Level 5:  Operations on terms
412
413  (define (multiply-terms term1 term2)          ;; Multiplies two terms
414    (make-term (+ (expon term1) (expon term2))  ;; and returns the resulting
415               (* (coef term1) (coef term2))))  ;; term.
416
417  (define (low-perf? term value tolerance)
418    (< (* (+ (expon term) value) (coef term)) tolerance))
419
420  (define (low-costbar? term tolerance)
421    (<= (* (coef term) (expon term)) tolerance))
422
423  (define (expon term)
424    (car term))
425
426  (define (coef term)
427    (cadr term))
428
429  (define (make-term expon coef)
430    (list expon coef))
431
432
433
434
435  ;; Utility Procedure for Obtaining Output
436
437  (define (print-list ls file-name per-line)
438    (let ((chan (open-printer-channel file-name)))
439      (let loop ((l ls) (count per-line))
440        (cond ((null? l) (close-channel chan))
441              ((zero? count)
442               (newline chan)
443               (loop l per-line))
444              (else
445               (princ (car l) chan)
446               (princ " " chan)
447               (loop (cdr l) (-1+ count)))))))
448
449
450  ;; Abbreviations for oft-used procedures
451
452  (define (mcm arg1 arg2 arg3 arg4) (multiply-col-mtx arg1 arg2 arg3 arg4))
453  (define (mc arg) (merge-column arg))
454  (define (ct arg) (count-terms arg))
455  (define (at arg) (add-elements arg))
456  (define (me arg) (merge-element arg))

457  (define (ac arg) (add-coef arg))
458  (define (mm arg1 arg2 arg3 arg4) (multiply-matrices arg1 arg2 arg3 arg4))
459  (define (exp arg1 arg2 arg3 arg4) (exponentiate arg1 arg2 arg3 arg4))
460  (define (prop arg1 arg2 arg3 arg4 arg5)
461    (gen-pmf-with-stats arg1 arg2 arg3 arg4 arg5))
```

```
;;  This is the file "FAST.SCM."

;; Top level (level 1) procedures:  Operations on matrices
   (declare (usual-integrations))

(define (prop-fast matrix steps begin-state vkm tol)
  (let loop ((result (make-init begin-state)) (count steps))
    (cond ((zero? count)
           (transform (add-elements result)))
          (else
           (print count)
           (set! result (merge-column (mult-col-mtx-fast result matrix
                                                         vkm tol)))
           (loop result (-1+ count))))))


(define (power matrix spec-list begin-state vkm tolerance)
  (let loop ((specs spec-list) (mtx matrix))
    (cond ((zero? (- (length specs) 1))
           (prop-fast mtx (car specs) begin-state (car specs)
                      vkm tolerance))
          (else
           (loop (cdr specs)
                 (exponentiate mtx (car specs) vkm tolerance))))))

(define (exponentiate matrix power vkm tolerance)
  (let loop ((result matrix) (count (-1+ power)))
    (cond ((zero? count)
           result)
          (else
           (print count)
           (set! result (multiply-matrices result matrix vkm tolerance))
           (loop result (-1+ count))))))

(define (multiply-matrices matrix1 matrix2 vkm tolerance)
  (let ((result '() ))
    (let loop ((matrix matrix1))
      (cond ((null? matrix)
             result)
            ((system-loss? (car matrix))
             (loop (cdr matrix)))
            (else
             (set! result (cons (merge-column (mult-col-mtx-fast (car matrix)
                                                                 matrix2
                                                                 vkm
                                                                 tolerance))
                                result))
             (loop (cdr matrix)))))))


;; Level 2: Operations on columns

(define (mult-col-mtx-fast column matrix vkm tol)   ;; Multiplies a column by a
  (let ((result '() ))                              ;; matrix and returns the elements
    (let loop1 ((els (elements column)))            ;; of the resulting column.
      (cond ((null? els)
             (make-column (column-number column) (cull-fast result vkm tol)))
            (else
             (let loop2 ((product (multiply-el-col (car els)
                                                   (get-column (row-number
                                                                (car els))
                                                               matrix))))
               (if (null? product)
                   (loop1 (cdr els))
                   (let ((element (assoc (row-number (car product))
                                         result)))
                     (cond ((null? element)
                            (set! result (cons (car product) result))
                            (loop2 (cdr product)))
                           (else
                            (set-cdr! element (list (append (transform element)
                                                            (transform
                                                             (car product)))))
                            (loop2 (cdr product)))))))))))

(define (merge-column column)                   ;; Returns column whose elements have
  (let ((result '() ))                          ;; had merge-element applied to them.
```

```
078       (let loop ((els (elements column)))
079         (cond ((null? els)
080                (make-column (column-number column) result))
081               (else
082                (set! result (cons (merge-element (car els))
083                                   result))
084                (loop (cdr els)))))))
085
086
087   ;; Level 3:  Operations on elements
088
089   (define (multiply-elements element1 element2)
090     (let ((result '() ))
091       (let loop1 ((tf1 (transform element1)) (tf2 (transform element2)))
092         (cond ((null? tf1)
093                (make-element (row-number element2) result))
094               (else                                      ;; Multiplies two elements
095                (let loop2 ((t2 tf2))                     ;; and returns the resulting
096                  (cond ((null? t2)                       ;; element, which takes the
097                         (loop1 (cdr tf1) tf2))           ;; row number of element2.
098                        (else
099                         (set! result (cons (multiply-terms (car tf1) (car t2))
100                                            result))
101                         (loop2 (cdr t2)))))))))))
102
103   (define (multiply-el-col element column)        ;; Multiplies the transform of
104     (let ((result '() ))                          ;; one element by the transforms
105       (let loop ((els (elements column)))         ;; of the elements of the column
106         (cond ((null? els)                        ;; and returns the list of
107                result)                            ;; resulting elements.
108               ((system-loss? (car els))
109                (loop (cdr els)))
110               (else
111                (set! result (cons (multiply-elements element (car els))
112                                   result))
113                (loop (cdr els)))))))
114
115
116   (define (cull-fast el-list vkm tol)
117     (let ((result '()))
118       (let loop ((els el-list))
119         (cond ((null? els)
120                result)
121               (else
122                (set! result (cons (make-element (row-number (car els))
123                                                 (cull-trans-fast
124                                                  (transform (car els))
125                                                  (cadr
126                                                   (assoc (row-number (car els))
127                                                          vkm))
128                                                  tol))
129                                   result))
130                (loop (cdr els)))))))
131
132
133
134   ;; Level 4:  Operations on transforms
135
136   (define (cull-trans-fast trnsform value tol)
137     (let ((result '()))
138       (let loop ((terms trnsform))
139         (cond ((null? terms)
140                result)
141               ((low-perf? (car terms) value tol)
142                (loop (cdr terms)))
143               (else
144                (set! result (cons (car terms) result))
145                (loop (cdr terms)))))))
146
147
```

```
001   ;; This file contains the procedures
      ;required to plot a performance PMF from
002   ;; the v-transform of that PMF.  The transform must be an alist indexed by
003   ;; cumulative performance values.  The top level procedure is "plot" which
004   ;; takes two arguments:  the transform and pts-per-bar.  Pts-per-bar allows
005   ;; a plot to be given a certain x-axis (the performance axis) scaling so that
006   ;; it is easier to compare two different PMFs.  If pts-per-bar is nil, the
007   ;; procedures will scale automatically.  The scaling of the y-axis (probability
008   ;; axis) is always logarythmic between 10**0 and 10**-15.  It is assumed that a
009   ;; tolerance will be selected during propagation which will cull any CSH that
010   ;; has a cumulative probability of less than 1e-15.
011   (declare (usual-integrations))
012
013   (define (plot transform pts-per-bar)
014     (let ((orgx -225) (orgy -150)
015           (rangex 450) (rangey 300)
016           (xint 20) (yint 20))
017       (draw-axes orgx orgy rangex rangey xint yint)
018       (draw-bars (process-y (process-x transform rangex orgx pts-per-bar)
019                             rangey
020                             orgy
021                             yint)
022                  orgy)))
023
024   (define (profile list yres plot-every)
025     (let ((orgx -225) (orgy -150))
026       (position-pen orgx orgy)
027       (let loop ((ls list) (counter plot-every) (k 1))
028         (cond ((null? ls)
029                'done)
030               ((eq? counter plot-every)
031                (draw-line-to (+ k orgx)
032                              (+ (floor (/ (car ls) yres))
033                                 orgy))
034                (loop (cdr ls) 0 (1+ k)))
035               (else
036                (loop (cdr ls) (1+ counter) k))))))
037
038   (define (draw-axes orgx orgy rangex rangey xint yint)
039     (clear-graphics)
040     (position-pen orgx orgy)
041     (draw-line-to orgx (+ orgy rangey))
042     (position-pen orgx orgy)
043     (draw-line-to (+ orgx rangex) orgy)
044     (let xchits ((x-pos 0))
045       (cond ((> x-pos rangex)
046              '())
047             (else
048              (position-pen (+ orgx x-pos) (- orgy 5))
049              (draw-line-to (+ orgx x-pos) orgy)
050              (xchits (+ x-pos xint)))))
051     (let ychits ((y-pos 0))
052       (cond ((> y-pos rangey)
053              '())
054             (else
055              (position-pen (- orgx 5) (+ orgy y-pos))
056              (draw-line-to orgx (+ orgy y-pos))
057              (ychits (+ y-pos yint))))))
058
059   (define (axes) (draw-axes -225 -150 450 300 20 20))
060
061   (define (draw-bars list-of-pairs orgy)
062     (let loop ((ls list-of-pairs))
063       (cond ((null? ls)
064              '())
065             (else
066              (position-pen (car (first ls)) orgy)
067              (draw-line-to (car (first ls)) (cadr (first ls)))
068              (loop (cdr ls))))))
069
070   (define (process-x transform rangex orgx pts-per-bar)
071     (let ((result '()) (max (max-perf transform)))
072       (cond ((null? pts-per-bar)
073              (define vals-per-bar (ceiling (/ max rangex)))
074              (princ "  Points per bar: ")
075              (prinl vals-per-bar)
076              (princ "    Maximum performance value: ")
```

```
077             (prin1 max))
078            (else
079             (define vals-per-bar pts-per-bar)))
080       (cond ((eq? vals-per-bar 1)
081              (let shift ((tf transform))
082                (cond ((null? tf)
083                       result)
084                      (else
085                       (set! result (cons (list (+ (car (first tf)) orgx)
086                                                (cadr (first tf)))
087                                          result))
088                       (shift (cdr tf))))))
089             (else
090              (let loop ((tf transform))
091                (cond ((null? tf)
092                       result)
093                      (else
094                       (let ((newx (+ (floor (/ (car (car tf))
095                                                vals-per-bar))
096                                      orgx)))
097                         (let ((pair (assv newx result)))
098                           (cond ((null? pair)
099                                  (set! result (cons (list newx (cadr (car tf)))
100                                                     result))
101                                  (loop (cdr tf)))
102                                 (else
103                                  (set-cdr! pair (list (+ (cadr (car tf))
104                                                          (cadr pair))))
105                                  (loop (cdr tf))))))))))))))
106
107   (define (process-y transform rangey orgy yint)
108     (let ((result '()))
109       (let loop ((tf transform))
110         (cond ((null? tf)
111                result)
112               (else
113                (set! result (cons (list (car (car tf))
114                                         (+ (round (* (/ (log (cadr (car tf)))
115                                                         2.30258)
116                                                      yint)) rangey orgy))
117                                   result))
118                (loop (cdr tf)))))))
119
120   (define (max-perf transform)
121     (let loop ((tf transform) (maxval 0))
122       (cond ((null? tf)
123              maxval)
124             ((> (car (car tf)) maxval)
125              (loop (cdr tf) (car (car tf))))
126             (else
127              (loop (cdr tf) maxval)))))
128
129   (define (min-perf transform)
130     (let loop ((tf transform) (minval (caar transform)))
131       (cond ((null? tf)
132              minval)
133             ((< (caar tf) minval)
134              (loop (cdr tf) (caar tf)))
135             (else
136              (loop (cdr tf) minval)))))
137
138   (define (min-coef transform)
139     (let loop ((tf transform) (minval (second (car transform))))
140       (cond ((null? tf)
141              minval)
142             ((< (second (car tf)) minval)
143              (loop (cdr tf) (second (car tf))))
144             (else
145              (loop (cdr tf) minval)))))
146
147
148   (define (max-coef transform)
149     (let loop ((tf transform) (maxval (second (car transform))))
150       (cond ((null? tf)
151              maxval)
152             ((> (second (car tf)) maxval)
```

```
153             (loop (cdr tf) (second (car tf))))
154            (else
155             (loop (cdr tf) maxval)))))
156
157   (define (shiftx transform val)
158     (let ((result '()))
159       (let loop ((tf transform))
160         (cond ((null? tf)
161                result)
162               (else
163                (set! result (cons (list (- (caar tf) val)
164                                         (second (car tf)))
165                                   result))
166                (loop (cdr tf)))))))
167
168
```

```
001   ;; Listing of the file "EXPANDER.SCM
      ;."  These procedures take the matrices
002   ;; of v-transforms for two independent subsystems and construct the matrix
003   ;; of v-transforms for the composite subsystem.  If the input matrices have
004   ;; ranks N+1 and M+1, then the resulting matrix will have the rank N*M+1.
005   ;; This file requires procedures in the approx.scm file, so load approx first.
006
007   ;; Make-matrix is the top level procedure.  It takes as arguments two
008   ;; previously define matrices of v-transforms.  It returns the composite
009   ;; matrix.
010   (declare (usual-integrations))
011
012   (define (make-matrix mtx1 mtx2)
013     (let ((result '())
014           (name-list (make-names (gen-state-list mtx1 mtx2)))
015           (syst-loss-col '(system-loss ((system-loss ((0 1)))))))
016       (let loop ((nlist name-list))
017         (cond ((null? nlist)
018                (append (reverse result) (list syst-loss-col)))
019               (else
020                (set! result (cons (gen-column (caar nlist) mtx1 mtx2 name-list)
021                                   result))
022                (loop (cdr nlist)))))))
023
024   ;; Gen-column (generate-column) takes as arguments:
025   ;;  1. state:  a pair of state designators, the first from matrix1 and the
026   ;;             second from matrix2, which as a pair designates a composite
027   ;;             state.
028   ;;  2. mtx1:  the v-transform matrix for subsystem1
029   ;;  3. mtx2:  the v-transform matrix for subsystem2
030   ;;  4. name-list:  an association list (alist) which has as its key the
031   ;;                 possible composite state pairs which can be defined for the
032   ;;                 particular mtx1 and mtx2.  The entry under each key is a
033   ;;                 number which is the new name for the composite state.  This
034   ;;                 is so it is not necessary to always refer to composite
035   ;;                 states by their list designations.
036   ;; Gen-column returns the column which results when mtx1 and mtx2 are
037   ;; expanded according to the composite state designator "state."  The
038   ;; resulting column and its elements have the new names specified in name-list.
039
040
041   (define (gen-column state mtx1 mtx2 name-list)
042     (let ((result '())
043           (syst-loss-prob 0))
044       (let loop1 ((els1 (elements (get-column (car state) mtx1)))
045                   (els2 (elements (get-column (cadr state) mtx2))))
046         (cond ((null? els1)
047                (cond ((zero? syst-loss-prob)
048                       (make-column (second (assoc state name-list))
049                                    (reverse result)))
050                      (else
051                       (make-column (second (assoc state name-list))
052                                    (append (reverse result)
053                                            (list
054                                             (list 'system-loss
055                                                   (list
056                                                    (list 0 syst-loss-prob)))))))))
057               (else
058                (let ((element1 (car els1)))
059                  (let loop2 ((elms2 els2))
060                    (cond ((null? elms2)
061                           (loop1 (cdr els1) els2))
062                          (else
063                           (let ((element2 (car elms2)))
064                             (let ((name (get-name (row-number element1)
065                                                   (row-number element2)
066                                                   name-list))
067                                   (term (multiply-terms (car (transform element1))
068                                                         (car (transform element2)))))
069                               (cond ((eq? name 'system-loss)
070                                      (set! syst-loss-prob (+ (coef term) syst-loss-prob))
071                                      (loop2 (cdr elms2)))
072                                     (else
073                                      (set! result (cons (make-element name (list term))
074                                                         result))
075                                      (loop2 (cdr elms2))))))))))))))))
076
```

```
077  ;; Gen-state-list generates a list of all possible composite states which can
078  ;; be defined for the matrices mtx1 and mtx2.  The composite states are
079  ;; returned as pairs.
080
081  (define (gen-state-list mtx1 mtx2)
082    (let ((result '()))
083      (let loop1 ((mx1 mtx1) (mx2 mtx2))
084        (cond ((null? mx1)
085               (reverse result))
086              ((system-loss? (car mx1))
087               (loop1 (cdr mx1) mx2))
088              (else
089               (let loop2 ((m2 mx2))
090                 (cond ((null? m2)
091                        (loop1 (cdr mx1) mx2))
092                       ((system-loss? (car m2))
093                        (loop2 (cdr m2)))
094                       (else
095                        (setl result (cons (list (column-number (car mx1))
096                                                 (column-number (car m2)))
097                                           result))
098                        (loop2 (cdr m2)))))))))))
099
100  ;; Make-names takes as its argument the state-list which is produced by the
101  ;; procedure gen-state-list.  It returns an alist which has the composite
102  ;; states in state-list as keys and integers (assigned in ascending order)
103  ;; for the new names.
104
105  (define (make-names state-list)
106    (let ((result '()))
107      (let loop ((counter 1) (sl state-list))
108        (cond ((null? sl)
109               (reverse result))
110              (else
111               (setl result (cons (list (car sl) counter)
112                                  result))
113               (loop (1+ counter) (cdr sl)))))))
114
115  ;; Get-name does an alist look-up based on state1 and state2 (s1 and s2) to
116  ;; find the new name in name-list for the composite state (s1 s2).
117
118  (define (get-name s1 s2 name-list)
119    (cond ((or (eq? s1 'system-loss) (eq? s2 'system-loss))
120           'system-loss)
121          (else
122           (second (assoc (list s1 s2) name-list)))))
123
124
125
126
```

```
C MAIN PROGRAM:  PROFILE.FOR
C PERFORMS ALL CALCULATIONS NECESSARY FOR GENERATING A PERFORMANCE
C PROFILE FOR A LARGE SYSTEM WITH TWO INDEPENDENT SUBSYSTEMS.
C THE CODE DOES NOT LIKE SYSTEMS WITH REPEATED OR COMPLEX EIGENVALUES,
C SO CHECK FOR THESE CONDITIONS BEFORE RUNNING THE PROGRAM.  THIS IS
C DUE TO THE FACT THAT THE IMSL ROUTINE EIGRF CANNOT FIND LINEARLY
C INDEPENDENT EIGENVECTORS FOR A REPEATED EIGENVALUE, IF THEY EXIST.
C COMPLEX EIGENVALUES SHOULD BOT BE A PROBLEM FOR MOST LOWER
C TRIANGULAR MARKOV MODELS.
C AFTER COMPILING, LINK THIS PROGRAM USING THE COMMAND:
C      LINK PROFILE,IMSLIBS/LIB,PENPLOT2$/OPT
C AND THEN USE
C      RUN PROFILE
C TO RUN THE PROGRAM.
C PRIOR TO COMPILATION, SET THE PARAMETER S TO BE THE ORDER OF THE
C LARGE COMPOSITE SYSTEM, i.e., THE PRODUCT OF THE ORDERS OF THE TWO
C SUBSYSTEMS.
C IF ANALYSIS OF ONLY ONE SUBSYSTEM IS DESIRED, SET S TO BE THE ORDER OF THE
C SUBSYSTEM, AND ENTER THE IDENTITY SYSTEM (NULL.DAT) AS THE SECOND SUB-
C SYSTEM WHEN THE PROGRAM ASKS FOR IT.
C
        INTEGER S,PTS,SS
        PARAMETER (S=6, PTS=110, SS=20)
        INTEGER N,DECS,PPD,R(S),NP1,J,RN,I,IJOB,K,SPEC,M,COL1,COL2,DESTROW
        INTEGER IER,IDGT,NPTS,SIZE,NEWIND(S,2),NEWMTX(20,20),INDEX
        INTEGER R1(SS),R2(SS)
        REAL P1(SS,SS),P2(SS,SS),DPVAL,SL(S),ARRAY(3,PTS)
        REAL P(S,S),EL,KM,Q(S),WK(S*S+2*S),VINF(S),VKM(S),RATIO
        REAL TIME(PTS),W(S,S),QV(S),COEFS(S),V1K(PTS)
        REAL MODE(PTS,S),RLAM(2*S),RV(2*S*S),REALV(S,S)
        COMPLEX LAM(S),V(S,S)
        CHARACTER*10 RESP,INFILE1,INFILE2,OUTFILE,AFILE
        CHARACTER*40 XL,YL
        LOGICAL CMPLEX,LOWTRI,EIGVALS,COMPS
        REAL LAMKM(S),VLAMKM(S,S),COL(S),REL
        EQUIVALENCE (LAM(1),RLAM(1)), (V(1,1),RV(1))
C
C INITIALIZATION SECTION TO READ P1, P2, R1, R2, AND SOME PARAMETERS.  NOTE
C THAT THE Ps ARE STORED IN A SPECIAL SPACE SAVING MODE SINCE THEY WILL
C GENERALLY BE SPARSE MATRICES.
C
        PRINT*,'SPECIFY SYSTEM 1 FILE NAME:  '
        READ(5,2) INFILE1
        PRINT*,'SPECIFY SYSTEM 2 FILE NAME:  '
        READ(5,2) INFILE2
 2      FORMAT(A10)
C
C READ R1 AND P1 FROM SYSTEM 1 FILE.
C
        OPEN(UNIT=1,FILE=INFILE1,STATUS='OLD')
        READ(1,4) N,SPEC
 4      FORMAT(2I16)
        READ(1,6)  (R1(K),K=1,N)
 6      FORMAT(5I16)
        READ(1,8)  (WK(K),K=1,SPEC)
 8      FORMAT(5F16.10)
        CLOSE(UNIT=1,STATUS='SAVE')
        DO 12 I=1,N
           DO 10 J=1,N
              P1(I,J) = 0.0
```

```
 10         CONTINUE
 12      CONTINUE
C
C WRITE P1 VALUES FROM SPACE SAVING MODE (IN WK) TO FULL STORAGE MODE IN
C THE P1 ARRAY.
C
         J = 0
         DO 16 K=1,SPEC
            IF((WK(K) .GT. 1.0) .OR. (K .EQ. 1)) GO TO 14
            P1(RN,J) = WK(K)
            RN = RN + 1
            GO TO 16
 14         J = J + 1
            RN = WK(K)
 16      CONTINUE
C
C READ R2 AND P2 FROM SYSTEM 2 FILE.
C
         OPEN(UNIT=1,FILE=INFILE2,STATUS='OLD')
         READ(1,18) M,SPEC
 18      FORMAT(2I16)
         READ(1,20) (R2(K), K=1,M)
 20      FORMAT(5I16)
         READ(1,22) (WK(K), K=1,SPEC)
 22      FORMAT(5F16.10)
         CLOSE(UNIT=1,STATUS='SAVE')
         DO 26 I=1,M
            DO 24 J=1,M
               P2(I,J) = 0.0
 24         CONTINUE
 26      CONTINUE
C
C WRITE P2 FROM SPACE SAVING MODE (IN WK) TO FULL STORAGE MODE IN
C THE P2 ARRAY.
C
         J = 0
         DO 30 K=1,SPEC
            IF((WK(K) .GT. 1.0) .OR. (K .EQ. 1)) GO TO 28
            P2(RN,J) = WK(K)
            RN = RN + 1
            GO TO 30
 28         J = J + 1
            RN = WK(K)
 30      CONTINUE
C
         PRINT*,'MISSION TIME (REAL NUMBER):  '
         READ(5,32) KM
 32      FORMAT(F16.0)
         PRINT*,'NUMBER OF DECADES FOR PROFILE (TWO DIGIT INTEGER):  '
         READ(5,34) DECS
         PRINT*,'POINTS PER DECADE (TWO DIGIT INTEGER):  '
         READ(5,34) PPD
 34      FORMAT(I2)
C
C INITIALIZE NEWIND AND NEWMTX MATRICES, WHICH ACT AS INDIRECT
C ARRAY SUBSCRIPTS FOR THE EXPANSION OF P1 AND P2 INTO P.
C
         DO 38 I=1,N
            DO 37 J=1,M
               NEWIND((I-1)*M+J,1) = I
```

```
                NEWIND((I-1)*M+J,2) = J
                NEWMTX(I,J) = (I-1)*M+J
 37          CONTINUE
 38       CONTINUE
C
C EXPAND P1 AND P2 INTO P
C
          SIZE = N * M
          DO 41 INDEX=1,SIZE
             COL1 = NEWIND(INDEX,1)
             COL2 = NEWIND(INDEX,2)
             DO 40 I=1,N
                DO 39 J=1,M
                   DESTROW = NEWMTX(I,J)
                   P(DESTROW,INDEX) = P1(I,COL1) * P2(J,COL2)
 39             CONTINUE
 40          CONTINUE
 41       CONTINUE
C
C COMPUTE THE SYSTEM LOSS TRANSITION PROBABILITY VECTOR
C
          DO 44 J=1,SIZE
             DPVAL = 0.0
             DO 43 I=1,SIZE
                DPVAL = DPVAL + P(I,J)
 43          CONTINUE
             SL(J) = 1.0 - DPVAL
 44       CONTINUE
C
C COMPUTE R FROM R1 AND R2
C
          DO 48 I=1,N
             DO 46 J=1,M
                INDEX = NEWMTX(I,J)
                R(INDEX) = R1(I) + R2(J)
 46          CONTINUE
 48       CONTINUE
C
C CHECK FOR A LOWER TRIANGULAR P MATRIX
C
          LOWTRI = .TRUE.
          DO 52 I=1,SIZE-1
             DO 50 J=I+1,SIZE
                IF(P(I,J) .NE. 0.0) LOWTRI = .FALSE.
 50          CONTINUE
 52       CONTINUE
          IF (LOWTRI) GO TO 54
          PRINT*,'P IS NOT LOWER TRIANGULAR, SKIPPING EIGVAL TEST.'
          GO TO 60
 54       CONTINUE
C
C CHECK FOR NO DUPLICATE EIGENVALUES
C
          EIGVALS = .TRUE.
          DO 58 I=1,SIZE-2
             DPVAL = P(I,I)
             DO 56 J=I+1,SIZE
                IF(DPVAL .EQ. P(J,J)) EIGVALS = .FALSE.
 56          CONTINUE
 58       CONTINUE
```

```
        IF (EIGVALS) GO TO 60
        PRINT*,'EXECUTION HALTING: REPEATED EVALS IN P.'
        PRINT*,'EIGENVALUES:'
        PRINT161, (P(I,I) ,I=1,SIZE)
        PRINT*,'P MATRIX BY ROWS:'
        PRINT161, ((P(I,J), J=1,SIZE), I=1,SIZE)
        STOP
 60     CONTINUE
C
C COMPUTATION OF EXPECTED IMMEDIATE REWARD (ROW) VECTOR, Q=RP.
C
        DO 90 J=1,SIZE
           Q(J)=0.0
           DO 80 I=1,SIZE
              Q(J) = Q(J) + R(I) * P(I,J)
 80        CONTINUE
 90     CONTINUE
        PRINT*,'Q'
C
C MODAL DECOMPOSITION OF P INTO V, LAM, AND W.
C
        IJOB = 2
        CALL EIGRF(P,S,S,IJOB,RLAM,RV,S,WK,IER)
        PRINT*,'EIGRF'
C OPTION TO HALT IF EIGRF PERFORMS POORLY.
        IF (WK(1) .LT. 100) GO TO 115
        WRITE(6,100) WK(1)
 100    FORMAT (' ','EIGRF PERF = ',F6.0,'.  SHOULD I PROCEED?')
        READ(5,110) RESP
 110    FORMAT(A1)
        IF (RESP .EQ. 'N') STOP
C COPY V TO PART OF REALV SINCE INPUT TO INV ROUTINE GETS DESTROYED.
        CMPLEX = .FALSE.
 115    DO 140 J=1,SIZE
           DO 130 I=1,SIZE
              REALV(I,J) = REAL(V(I,J))
              IF (AIMAG(V(I,J)) .EQ. 0.0) GO TO 130
              WRITE(6,120) J
 120          FORMAT(' ','EIGENVECTOR ',I3,' IS COMPLEX.')
              CMPLEX = .TRUE.
 130       CONTINUE
 140    CONTINUE
        PRINT*,'V COPIED'
        IDGT = 7
        IF (CMPLEX) STOP
        CALL LINV1F(REALV,S,S,W,IDGT,WK,IER)
        PRINT*,'LINV1F'
C OPTION TO HALT IF LINV1F PERFORMS POORLY.
        IF (IER .NE. 129) GO TO 168
        WRITE(6,150) IER
 150    FORMAT (' ','LINV1F IER = ',I3,'.  CHECK THE S PARAM OR EVALS.',
     1          '  SHOULD I PROCEED?')
        READ(5,160) RESP
 160    FORMAT(A1)
        IF (RESP .NE. 'N') GO TO 168
        PRINT161,((REAL(V(I,J)), J=1,SIZE),I=1,SIZE)
        PRINT161,((W(I,J),J=1,SIZE),I=1,SIZE)
 161    FORMAT(6(1X,G12.6))
        STOP
C
```

```
C COMPUTE RELIABILITY AT MISSION TIME.
C
 168     DO 162 I=1,SIZE
            LAMKM(I) = REAL(LAM(I))**KM
 162     CONTINUE
         DO 164 J=1,SIZE
            DO 163 I=1,SIZE
               VLAMKM(I,J) = REAL(V(I,J)) * LAMKM(J)
 163        CONTINUE
 164     CONTINUE
         DO 166 I=1,SIZE
            COL(I) = 0.0
            DO 165 J=1,SIZE
               COL(I) = COL(I) + VLAMKM(I,J) * W(J,1)
 165        CONTINUE
 166     CONTINUE
         REL = 0.0
         DO 167 I=1,SIZE
            REL = REL + COL(I)
 167     CONTINUE
C
C COMPUTATION OF QV
C
 169     DO 180 J=1,SIZE
            QV(J) = 0.0
            DO 170 I=1,SIZE
               QV(J) = QV(J) + Q(I) * REAL(V(I,J))
 170        CONTINUE
 180     CONTINUE
         PRINT*,'QV'
C
C COMPUTATION OF THE INFINITE HORIZON TOTAL VALUE VECTOR, VINF.
C
         DO 190 J=1,SIZE
            WK(J) = QV(J)/(1 - REAL(LAM(J)))
 190     CONTINUE
         DO 210 J=1,SIZE
            VINF(J) = 0.0
            DO 200 I=1,SIZE
               VINF(J) = VINF(J) + WK(I) * W(I,J)
 200        CONTINUE
 210     CONTINUE
         PRINT*,'VINF'
C
C COMPUTATION OF THE TOTAL VALUE VECTOR AT MISSION TIME, VKM.
C
         DO 220 J=1,SIZE
            WK(J) = QV(J) * (REAL(LAM(J)) ** KM - 1) / (REAL(LAM(J)) - 1)
 220     CONTINUE
         DO 240 J=1,SIZE
            VKM(J) = 0.0
            DO 230 I=1,SIZE
               VKM(J) = VKM(J) + WK(I) * W(I,J)
 230        CONTINUE
 240     CONTINUE
         PRINT*,'VKM'
C
C COMPUTATION OF MODAL COEFFICIENTS FOR V1(K).
C
         DO 250 I=1,SIZE
```

```
            COEFS(I) = QV(I) * W(I,1)
 250      CONTINUE
          PRINT*,'COEFS'
C
C COMPUTATION OF VALUES FOR MODAL PLOT OF PERFORMANCE PROFILE
C
          RATIO = 10.0E0 ** (1.0E0 / PPD)
          DO 270 I=1,DECS
             DO 260 J=1,PPD
                TIME((I-1) * PPD + J) = INT(10.0 ** (I - 1) * RATIO ** (J - 1))
 260         CONTINUE
 270      CONTINUE
          PRINT*,'TIME'
          NPTS = DECS * PPD
          DO 290 I=1,NPTS
             V1K(I) = 0.0
             DO 280 J=1,SIZE
                MODE(I,J)=COEFS(J)*(REAL(LAM(J))**TIME(I) - 1)/(REAL(LAM(J)) - 1)
                V1K(I) = V1K(I) + MODE(I,J)
 280         CONTINUE
 290      CONTINUE
          PRINT*,'MODE'
C
C SELECTIVELY SAVE INFORMATION IN OUTFILE.
C
          PRINT*,'SAVE ANY INFO?'
          READ(5,510) RESP
          IF(RESP .EQ. 'N') GO TO 520
          PRINT*,'SPECIFY OUTPUT FILE NAME:'
          READ(5,2) OUTFILE
          OPEN(UNIT=1,FILE=OUTFILE,STATUS='NEW')
          WRITE(1,295) OUTFILE,INFILE1,INFILE2
 295      FORMAT(' ','PRINTOUT OF OUTPUT FILE ',A10
     1           /' ','SYSTEM 1 INPUT FILE WAS ',A10
     2           /' ','SYSTEM 2 INPUT FILE WAS ',A10)
          WRITE(1,300) KM
 300      FORMAT(' ','MISSION TIME K = ',F16.0)
          WRITE(1,310)
 310      FORMAT(' ','TOTAL VALUE VECTOR AT MISSION TIME:')
          WRITE(1,325) (VKM(K), K=1,SIZE)
          WRITE(1,315) REL
 315      FORMAT(' ','RELIABILITY AT MISSION TIME:  ',F16.10)
          WRITE(1,320)
 320      FORMAT(' ','TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:')
          WRITE(1,325) (VINF(K), K=1,SIZE)
 325      FORMAT(' ',5F16.4)
C
          PRINT*,'SAVE EIGENVALUES?'
          READ(5,510) RESP
          IF(RESP .EQ. 'N') GO TO 340
          WRITE(1,330)
 330      FORMAT(' ','EIGENVALUES OF STATE TRANSITION MATRIX:')
          WRITE(1,360) (REAL(LAM(K)), K=1,SIZE)
C
 340      CONTINUE
          PRINT*,'SAVE MODAL COEFFICIENTS OF V1(K)?'
          READ(5,510) RESP
          IF(RESP .EQ. 'N') GO TO 370
          WRITE(1,350)
 350      FORMAT(' ','MODAL COEFFICIENTS IN EXPANSION OF V1(K):')
```

```
      WRITE(1,360) (COEFS(K), K=1,SIZE)
360   FORMAT(' ',5F16.10)
C
370   CONTINUE
      PRINT*,'SAVE R VECTOR AND P MATRIX?'
      READ(5,510) RESP
      IF(RESP .EQ. 'N') GO TO 410
      WRITE(1,380)
380   FORMAT(' ','PERFORMANCE VECTOR:')
      WRITE(1,390) (R(K), K=1,SIZE)
390   FORMAT(' ',5I16)
      IF (LOWTRI) GO TO 402
      WRITE(1,400)
400   FORMAT('0','FULL STATE TRANSITION MATRIX LISTED BY COLUMNS:')
      WRITE(1,440) ((P(I,J), I=1,SIZE), J=1,SIZE)
      GO TO 408
402   WRITE(1,404)
404   FORMAT('0','CONCISE LOW TRI STATE TRANS MATRIX BY COLUMNS:')
      DO 408 J=1,SIZE
         WRITE(1,405) J
405      FORMAT(' ','COLUMN ',I2,':')
         WRITE(1,406) (P(I,J),I=J,SIZE)
406      FORMAT(' ',5F16.10)
408   CONTINUE
      WRITE(1,409)
409   FORMAT('0','SYSTEM LOSS TRANSITION PROBABILITY VECTOR:')
      WRITE(1,440) (SL(K), K=1,SIZE)
C
410   CONTINUE
      PRINT*,'SAVE MATRICES OF RIGHT AND LEFT EIGENVECTORS?'
      READ(5,510) RESP
      IF(RESP .EQ. 'N') GO TO 450
      WRITE(1,420)
420   FORMAT('0','MATRIX OF RIGHT EIGENVECTORS (LISTED BY COLUMNS):')
      WRITE(1,440) ((REALV(I,J), I=1,SIZE), J=1,SIZE)
      WRITE(1,430)
430   FORMAT('0','MATRIX OF LEFT EIGENVECTORS (LISTED BY COLUMNS):')
      WRITE(1,440) ((W(I,J), I=1,SIZE), J=1,SIZE)
440   FORMAT(' ',5F16.10)
C
450   CONTINUE
      PRINT*,'SAVE V1(K) PROFILE VALUES?'
      READ(5,510) RESP
      IF(RESP .EQ. 'N') GO TO 520
      PRINT*,'SAVE MODAL COMPONENTS OF V1(K) TOO?'
      READ(5,510) RESP
      COMPS = .FALSE.
      IF(RESP .NE. 'N') COMPS = .TRUE.
      WRITE(1,460) DECS
460   FORMAT('0','EXPECTED PERFORMANCE PROFILE FOR K = 1 TO 10**',
     1       I2)
      DO 500 K=1,NPTS
         WRITE(1,470) TIME(K)
         IF(COMPS) WRITE(1,480) (MODE(K,J), J=1,SIZE)
         WRITE(1,490) V1K(K)
470      FORMAT('0','FOR K = ',F16.0)
480      FORMAT(' ',5F16.10)
490      FORMAT(' ','V1(K) = ',F16.4)
500   CONTINUE
510   FORMAT(A1)
```

```
      CLOSE(UNIT=1,STATUS='SAVE')
 520  CONTINUE
C
      PRINT*,'GENERATE A PLOT OF THE EXPECTED PERFORMANCE?'
      READ(5,510) RESP
      IF (RESP .EQ. 'N') GO TO 523
C
C GENERATE PLOT OF EXPECTED PERFORMANCE PROFILE
C
C FIRST COPY TIME, V1K, AND VINF(1) TO ARRAY.
C
      DO 522 I=1,NPTS
         ARRAY(1,I) = TIME(I)
         ARRAY(2,I) = V1K(I)
         ARRAY(3,I) = VINF(1)
 522  CONTINUE
      XL = 'TIME K'
      YL = 'EXPECTED PERFORMANCE, E[J]'
      CALL QPICTR(ARRAY,3,NPTS,QXLAB(XL),QYLAB(YL),QLABEL(4),
     1            QISCL(1),QX(1),QY(2,3))
 523  PRINT*,'SAVE PLOT ARRAY FOR FUTURE PLOTTING?'
      READ(5,510) RESP
      IF (RESP .EQ. 'N') GO TO 530
      PRINT*,'SPECIFY ARRAY FILE NAME:'
      READ(5,2) AFILE
      OPEN(UNIT=1,FILE=AFILE,STATUS='NEW')
      WRITE(1,525) ((ARRAY(I,J),J=1,NPTS),I=1,3)
 525  FORMAT(5F16.4)
      CLOSE(UNIT=1,STATUS='SAVE')
 530  CONTINUE
      STOP
      END
```

APPENDIX C

FORTRAN RESULTS

```
PRINTOUT OF OUTPUT FILE FOR 7 STATE MODEL
SYSTEM 1 INPUT FILE WAS STATE7.DAT
SYSTEM 2 INPUT FILE WAS NULL.DAT
MISSION TIME K =                150.
TOTAL VALUE VECTOR AT MISSION TIME:
         62.7363         101.1588         102.4084          58.0350          57.004
         56.9740
RELIABILITY AT MISSION TIME:       0.4700905979
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
        110.2767         108.0375         109.5156          58.0610          57.030
         57.0000
EIGENVALUES OF STATE TRANSITION MATRIX:
     0.9499999881     0.0299999993     0.9800000191     0.0399999991     0.050000000
     0.9900000095
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
     0.4552075863    -0.0068692365    -2.9027307034     0.0195787884    -0.013283506
     2.4630970955


PRINTOUT OF OUTPUT FILE FOR 50 STATE MODEL
SYSTEM 1 INPUT FILE WAS SENSORS.DAT
SYSTEM 2 INPUT FILE WAS ACTUATORS.DAT
MISSION TIME K =                7200.
TOTAL VALUE VECTOR AT MISSION TIME:
      101296.7344      143869.7031      143820.0156      143882.7188      142666.984
      138798.9844      166291.8125      102511.7266      141673.8750      141578.281
      141714.7344      139484.1406      135702.3906      162580.9688      102266.562
      141263.5156      141166.7188      141305.4375      139041.7969      135272.046
      162065.2656      102659.3672      141921.8281      141827.2500      141961.937
      139751.6719      135962.5469      162892.8594       97118.4453      132647.359
      132518.2031      132710.1719      129750.3359      126232.6016      151236.484
       94289.5000      128783.4766      128658.2188      128844.5547      125970.882
      122555.6797      146831.3125      113143.5000      154534.6094      154383.906
      154608.8438      151158.8125      147061.2031      176192.2344
RELIABILITY AT MISSION TIME:       0.9796848893
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
     1262333.3750     1071391.8750     1061907.5000     1077135.2500      867125.875
      843662.2500     1010926.7500     1040346.6250      899014.7500      891246.625
      903717.7500      731679.5625      711880.3125      853012.6250     1039759.000
      898255.3750      890489.8125      902957.8125      730952.0625      711172.687
      852164.4375     1040697.1875      899471.6875      891702.7500      904173.187
      732117.3125      712305.5625      853522.6250     1027419.6250      882310.625
      874573.5625      886994.3125      715667.1250      696301.0625      834347.500
      997494.4375      856611.3750      849100.2500      861159.0000      694821.625
      676020.1250      810045.7500     1196980.8750     1027921.5625     1018907.437
     1033379.2500      833774.6250      811213.4375      972043.4375
EIGENVALUES OF STATE TRANSITION MATRIX:
     0.9999722242     0.2799937725     0.0799982175     0.9998190403     0.299993336
     0.0499988906     0.9999616742     0.2809984386     0.0786799937     0.022479997
     0.2809553742     0.0842999965     0.0140499994     0.2809954584     0.078999564
     0.0221200008     0.0063200002     0.0789874643     0.0237000026     0.003950000
     0.0789987296     0.9998361468     0.2799556851     0.0799873322     0.999682962
     0.2999525070     0.0499920845     0.9998255968     0.2989983261     0.083719998
     0.0239199996     0.2989525497     0.0896999985     0.0149499997     0.298995196
     0.0498997234     0.0139720002     0.0039919997     0.0498920791     0.014969999
     0.0024949999     0.0498991944     0.9999449849     0.2799861431     0.079996042
     0.9997918010     0.2999851704     0.0499975272     0.9999344349
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
    72.4428634644     0.0000523517    -0.0000002238    -0.6114380956    -0.000062723
     0.0000000088   -20.9098968506     0.0011715272     0.0000000005     0.000000000
    -0.0000133974    -0.0000000005     0.0000000000    -0.0002682131     0.000022682
     0.0000000000     0.0000000000    -0.0000002633     0.0000000000     0.000000000
    -0.0000051608     6.1132912636     0.0000030365    -0.0000000150    -0.064252421
    -0.0000034946     0.0000000009    -1.5208215714    -0.0016868892    -0.000000000
     0.0000000000     0.0000189747     0.0000000008     0.0000000000     0.000403281
    -0.0000052534     0.0000000000     0.0000000000     0.0000000611     0.000000000
     0.0000000000     0.0000011832   -58.7966423035    -0.0000357349     0.000000162
     0.5578575134     0.0000421797    -0.0000000079    15.7902603149
```

```
PRINTOUT OF OUTPUT FILE FOR 10 STATE MODEL
SYSTEM 1 INPUT FILE WAS SENS4.DAT
SYSTEM 2 INPUT FILE WAS ACTS4.DAT
MISSION TIME K =              7200.
TOTAL VALUE VECTOR AT MISSION TIME:
     100081.7266     143873.1875     164129.1719     110722.2734     154739.031
     174599.1406     113377.0938     156708.3906     176192.2344
RELIABILITY AT MISSION TIME:       0.9617496133
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
    1206475.3750    1041040.7500     954063.9375    1259931.5000    1087796.375
     996130.3125    1225833.6250    1060561.6250     972043.4375
EIGENVALUES OF STATE TRANSITION MATRIX:
    0.9999722242    0.9998190403    0.9999616742    0.9998360276    0.999682843
    0.9998254776    0.9999449849    0.9997918010    0.9999344349
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
   64.9560089111    0.3387398720  -12.5435619354    5.8889918327    0.021143509
   -0.8916142583  -53.7432746887   -0.2342728227    9.2080230713
PERFORMANCE VECTOR:
              13              18              25              14              19
              26              15              20              27
CONCISE LOW TRI STATE TRANS MATRIX BY COLUMNS:
COLUMN  1:
    0.9999344349    0.0000494442    0.0000000000    0.0000116105    0.000000000
    0.0000000000    0.0000000000    0.0000000000    0.0000000000
COLUMN  2:
    0.9998254776    0.0001583265    0.0000000000    0.0000116093    0.000000001
    0.0000000000    0.0000000000    0.0000000000
COLUMN  3:
    0.9999616742    0.0000000000    0.0000000000    0.0000116108    0.000000000
    0.0000000000    0.0000000000
COLUMN  4:
    0.9997918010    0.0000494372    0.0000000000    0.0001586921    0.000000007
    0.0000000000
COLUMN  5:
    0.9996828437    0.0001583039    0.0000000000    0.0001586749    0.000000025
COLUMN  6:
    0.9998190403    0.0000000000    0.0000000000    0.0001586965
COLUMN  7:
    0.9999449849    0.0000494447    0.0000000000
COLUMN  8:
    0.9998360276    0.0001583281
COLUMN  9:
    0.9999722242
SYSTEM LOSS TRANSITION PROBABILITY VECTOR:
    0.0000044703    0.0000045896    0.0000267029    0.0000001192    0.00000017
    0.0000222921    0.0000055432    0.0000056624    0.0000277758
```

```
PRINTOUT OF OUTPUT FILE FOR CASE 1
SYSTEM 1 INPUT FILE WAS CASE1.DAT
SYSTEM 2 INPUT FILE WAS NULL.DAT
MISSION TIME K =            7200.
TOTAL VALUE VECTOR AT MISSION TIME:
       7661.4551      15810.6689      15918.6836      15744.1455      18176.919
      17647.3516      21174.5586
RELIABILITY AT MISSION TIME:      0.9918781519
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
      525863.6250     463226.2813     463286.5313     463186.2500     464533.437
      451003.1250     541197.5000
EIGENVALUES OF STATE TRANSITION MATRIX:
    0.9999944568    0.2800000012    0.0799999982    0.9998412728    0.300000011
    0.0500000007    0.9999839067
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
    3.9519212246    0.0000136059   -0.0000000422    0.0651749074   -0.000017300
   -0.0000000006   -3.0170705318
PERFORMANCE VECTOR:
               1               4               4               2               7
               7               3
CONCISE LOW TRI STATE TRANS MATRIX BY COLUMNS:
COLUMN  1:
    0.9999839067    0.0000001111    0.0000026400    0.0000132499    0.000000110
    0.0000000000    0.0000000000
COLUMN  2:
    0.0500000007    0.2280000001    0.7124999762    0.0094999997    0.000000000
    0.0000000000
COLUMN  3:
    0.3000000119    0.6499999762    0.0500000007    0.0000000000    0.000000000
COLUMN  4:
    0.9998412728    0.0000001667    0.0000020417    0.0001303072
COLUMN  5:
    0.0799999982    0.2300000042    0.5979999900
COLUMN  6:
    0.2800000012    0.6000000238
COLUMN  7:
    0.9999944568
SYSTEM LOSS TRANSITION PROBABILITY VECTOR:
    0.0000000000    0.0000000000    0.0000000000    0.0000262260    0.092000007
    0.1199999452    0.0000055432
```

```
PRINTOUT OF OUTPUT FILE FOR CASE 2
SYSTEM 1 INPUT FILE WAS CASE2.DAT
SYSTEM 2 INPUT FILE WAS NULL.DAT
MISSION TIME K =            7200.
TOTAL VALUE VECTOR AT MISSION TIME:
       8404.9609      18105.2070      18108.7539      18103.3457      18176.919
      17647.3516      21174.5586
RELIABILITY AT MISSION TIME:      0.9971009493
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
     573334.5625     544256.5625     540628.3125     546477.1250     464533.437
     451003.1250     541197.5000
EIGENVALUES OF STATE TRANSITION MATRIX:
    0.9999944568    0.2800000012    0.0799999982    0.9997416735    0.300000011
    0.0500000007    0.9999638796
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
    3.5994093418    0.0000136064   -0.0000000422    0.1694644690   -0.000017301
   -0.0000000006   -2.7688279152
PERFORMANCE VECTOR:
               1               4               4               2               7
               7               3
CONCISE LOW TRI STATE TRANS MATRIX BY COLUMNS:
COLUMN  1:
    0.9999638796    0.0000001111    0.0000026400    0.0000332497    0.000000110
    0.0000000000    0.0000000000
COLUMN  2:
    0.0500000007    0.2280000001    0.7124999762    0.0094999997    0.000000000
    0.0000000000
COLUMN  3:
    0.3000000119    0.6499999762    0.0500000007    0.0000000000    0.000000000
COLUMN  4:
    0.9997416735    0.0000001667    0.0000020417    0.0002553062
COLUMN  5:
    0.0799999982    0.2300000042    0.5979999900
COLUMN  6:
    0.2800000012    0.6000000238
COLUMN  7:
    0.9999944568
SYSTEM LOSS TRANSITION PROBABILITY VECTOR:
    0.0000000000    0.0000000000    0.0000000000    0.0000008345    0.092000007
    0.1199999452    0.0000055432
```

```
PRINTOUT OF OUTPUT FILE FOR CASE 3
SYSTEM 1 INPUT FILE WAS CASE3.DAT
SYSTEM 2 INPUT FILE WAS NULL.DAT
MISSION TIME K =              7200.
TOTAL VALUE VECTOR AT MISSION TIME:
      14620.5732      20739.6875      20623.2891      20811.4844      18176.919
      17647.3516      21174.5586
RELIABILITY AT MISSION TIME:      0.9797978997
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
     545559.8750     539642.8750     536224.3125     541734.5625     464533.437
     451003.1250     541197.5000
EIGENVALUES OF STATE TRANSITION MATRIX:
    0.9999944568    0.2800000012    0.0799999982    0.9974916577    0.300000011
    0.0500000007    0.9997389317
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
    3.0703322887    0.0000136119   -0.0000000422    0.1163672656   -0.000017310
   -0.0000000006   -2.1864285469
PERFORMANCE VECTOR:
               1               4               4               2               7
               7               3
CONCISE LOW TRI STATE TRANS MATRIX BY COLUMNS:
COLUMN  1:
    0.9997389317    0.0000001111    0.0000026400    0.0002582472    0.000000110
    0.0000000000    0.0000000000
COLUMN  2:
    0.0500000007    0.2280000001    0.7124999762    0.0094999997    0.000000000
    0.0000000000
COLUMN  3:
    0.3000000119    0.6499999762    0.0500000007    0.0000000000    0.000000000
COLUMN  4:
    0.9974916577    0.0000001667    0.0000020417    0.0025052871
COLUMN  5:
    0.0799999982    0.2300000042    0.5979999900
COLUMN  6:
    0.2800000012    0.6000000238
COLUMN  7:
    0.9999944568
SYSTEM LOSS TRANSITION PROBABILITY VECTOR:
    0.0000000000    0.0000000000    0.0000000000    0.0000008345    0.092000007
    0.1199999452    0.0000055432
```

```
PRINTOUT OF OUTPUT FILE FOR CASE 4
SYSTEM 1 INPUT FILE WAS CASE4.DAT
SYSTEM 2 INPUT FILE WAS NULL.DAT
MISSION TIME K =              7200.
TOTAL VALUE VECTOR AT MISSION TIME:
       9995.1602      15951.5039      15921.4404      15969.4111      15283.525
      14838.2051      17802.3730
RELIABILITY AT MISSION TIME:      0.9388317466
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
      68791.2578      60163.0273      59535.0703      60544.4180      46356.394
      45005.9844      54000.9141
EIGENVALUES OF STATE TRANSITION MATRIX:
    0.9999444485    0.2800000012    0.0799999982    0.9997916818    0.300000011
    0.0500000007    0.9998838902
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
    7.3351488113    0.0001360330   -0.0000004218    2.2798321247   -0.000172978
   -0.0000000063   -8.6147670746
PERFORMANCE VECTOR:
               1               4               4               2               7
               7               3
CONCISE LOW TRI STATE TRANS MATRIX BY COLUMNS:
COLUMN  1:
    0.9998838902    0.0000011111    0.0000264000    0.0000874994    0.000001100
    0.0000000000    0.0000000000
COLUMN  2:
    0.0500000007    0.2280000001    0.7124999762    0.0094999997    0.000000000
    0.0000000000
COLUMN  3:
    0.3000000119    0.6499999762    0.0500000007    0.0000000000    0.000000000
COLUMN  4:
    0.9997916818    0.0000016667    0.0000204167    0.0001780729
COLUMN  5:
    0.0799999982    0.2300000042    0.5979999900
COLUMN  6:
    0.2800000012    0.6000000238
COLUMN  7:
    0.9999444485
SYSTEM LOSS TRANSITION PROBABILITY VECTOR:
    0.0000000000    0.0000000000    0.0000000000    0.0000081062    0.092000007
    0.1199999452    0.0000555515
```

```
PRINTOUT OF OUTPUT FILE FOR CASE 5
SYSTEM 1 INPUT FILE WAS CASE5.DAT
SYSTEM 2 INPUT FILE WAS NULL.DAT
MISSION TIME K =              7200.
TOTAL VALUE VECTOR AT MISSION TIME:
      10584.5488      16556.1563      16498.5645      16590.9648      15283.525
      14838.2051      17802.3730
RELIABILITY AT MISSION TIME:       0.9186308980
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
      65166.3711      57775.9727      57256.6797      58090.7617      46356.394
      45005.9844      54000.9141
EIGENVALUES OF STATE TRANSITION MATRIX:
    0.9999444485    0.2800000012    0.0799999982    0.9996666908    0.300000011
    0.0500000007    0.9998638630
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
    5.8312849998    0.0001360379   -0.0000004218    0.9936411977   -0.000172985
   -0.0000000063   -5.8246917725
PERFORMANCE VECTOR:
               1               4               4               2               7
               7               3
CONCISE LOW TRI STATE TRANS MATRIX BY COLUMNS:
COLUMN  1:
    0.9998638630    0.0000011111    0.0000264000    0.0001074972    0.000001100
    0.0000000000    0.0000000000
COLUMN  2:
    0.0500000007    0.2280000001    0.7124999762    0.0094999997    0.000000000
    0.0000000000
COLUMN  3:
    0.3000000119    0.6499999762    0.0500000007    0.0000000000    0.000000000
COLUMN  4:
    0.9996666908    0.0000016667    0.0000204167    0.0003030625
COLUMN  5:
    0.0799999982    0.2300000042    0.5979999900
COLUMN  6:
    0.2800000012    0.6000000238
COLUMN  7:
    0.9999444485
SYSTEM LOSS TRANSITION PROBABILITY VECTOR:
    0.0000000000    0.0000000000    0.0000000000    0.0000081062    0.092000007
    0.1199999452    0.0000555515
```

```
PRINTOUT OF OUTPUT FILE FOR CASE 6
SYSTEM 1 INPUT FILE WAS CASE6.DAT
SYSTEM 2 INPUT FILE WAS NULL.DAT
MISSION TIME K =            7200.
TOTAL VALUE VECTOR AT MISSION TIME:
      14337.6641      17639.4141      17532.5137      17704.2363      15283.525
      14838.2051      17802.3730
RELIABILITY AT MISSION TIME:      0.7897561193
TOTAL VALUE VECTOR FOR INFINITE TIME HORIZON:
      57231.3984      54310.8984      53949.3203      54529.0000      46356.394
      45005.9844      54000.9141
EIGENVALUES OF STATE TRANSITION MATRIX:
    0.9999444485    0.2800000012    0.0799999982    0.9974168539    0.300000011
    0.0500000007    0.9996389151
MODAL COEFFICIENTS IN EXPANSION OF V1(K):
    3.6018893719    0.0001360926   -0.0000004220    0.1690896899   -0.000173076
   -0.0000000063   -2.7705199718
PERFORMANCE VECTOR:
               1               4               4               2               7
               7               3
CONCISE LOW TRI STATE TRANS MATRIX BY COLUMNS:
COLUMN  1:
    0.9996389151    0.0000011111    0.0000264000    0.0003324722    0.000001100
    0.0000000000    0.0000000000
COLUMN  2:
    0.0500000007    0.2280000001    0.7124999762    0.0094999997    0.000000000
    0.0000000000
COLUMN  3:
    0.3000000119    0.6499999762    0.0500000007    0.0000000000    0.000000000
COLUMN  4:
    0.9974168539    0.0000016667    0.0000204167    0.0025528751
COLUMN  5:
    0.0799999982    0.2300000042    0.5979999900
COLUMN  6:
    0.2800000012    0.6000000238
COLUMN  7:
    0.9999444485
SYSTEM LOSS TRANSITION PROBABILITY VECTOR:
    0.0000000000    0.0000000000    0.0000000000    0.0000081658    0.092000007
    0.1199999452    0.0000555515
```

REFERENCES

1. Abelson, H., and Sussman, G. J., with Sussman, J., Structure and Interpretation of Computer Programs, M.I.T. Press, Cambridge, MA, 1985.

2. Carignan, C. R., "Filter Failure Detection for Systems with Large Space Structure Dynamics," S.M. Thesis, Dept. of Aeronautics and Astronautics, M.I.T., Cambridge, MA, June 1982.

3. Carignan, C. R., and VanderVelde, W. E., "Number and Placement of Control System Components Considering Possible Failures," M.I.T. Space Systems Lab., Report 5-82, M.I.T., Cambridge, MA, March 1982.

4. Gai, E., and Adams, M., "Measures of Merit for Fault-Tolerant Systems," C. S. Draper Laboratory, Cambridge, MA, Report P-1752, 1983.

5. Harrison, J. V., Daly, K. C., and Gai, E., "Reliability and Accuracy Prediction for a Redundant Strapdown Navigator," Journal of Guidance and Control, 4:5: pp. 523-529, September-October 1981.

6. Howard, R., Dynamic Programming and Markov Processes, M.I.T. Press, Cambridge, MA, 1960, pp. 18-24.

7. Luppold, R. H., "Reliability and Availability Models for Fault-Tolerant Systems," S.M. Thesis, Dept. of Aeronautics and Astronautics, M.I.T., Cambridge, MA, August 1983.

8. Major, C. S., "A Demonstration of the Use of Generalized Parity Relations for Detection and Identification of Instrumentation Failures on a Free-free Beam," M.I.T., S.M. Thesis, Dept. of Aeronautics and Astronautics, M.I.T., Cambridge, MA, September 1981.

9. Meserole, J. S., "Detection filters for Fault-Tolerant Control of Turbofan Engines," Ph.D. Thesis, Dept. of Aeronautics and Astronautics, M.I.T., Cambridge, MA, June 1981.

10. VanderVelde, W. E., "Control System Reconfiguration," Dept. of Aeronautics and Astronautics, M.I.T., Cambridge, MA.

11. VanderVelde, W. E., and Carignan, C. R., "A Dynamic Measure of Controllability and Observability for the Placement of Actuators and Sensors on Large Space Structures," M.I.T. Space Systems Lab., Report 2-82, M.I.T., Cambridge, Massachusetts, January 1982.

12. Walker, B. K., "A Semi-Markov Approach to Quantifying Fault-Tolerant System Performance," Sc.D. Thesis, M.I.T. Dept. of Aeronautics and Astronautics, July 1980.

13. Walker, B. K., and Gai, E., "Fault Detection Threshold Technique Using Markov Theory," Journal of Guidance and Control, Vol. 2, July-August 1979, pp. 313 - 319.