RELIABILITY CALCULATION USING RANDOMIZATION FOR MARKOVIAN FAULT-TOLERANT COMPUTING SYSTEMS

by

Douglas R. Miller

Serial T-469
29 October 1982

The George Washington University
School of Engineering and Applied Science
Department of Operations Research
Institute for Management Science and Engineering

Research Supported by

Grant NAG-1-179
National Aeronautics and Space Administration

THE GEORGE WASHINGTON UNIVERSITY
School of Engineering and Applied Science
Department of Operations Research
Institute for Management Science and Engineering

RELIABILITY CALCULATION USING RANDOMIZATION FOR MARKOVIAN FAULT-TOLERANT COMPUTING SYSTEMS

by

Douglas R. Miller

1. INTRODUCTION

The behavior of fault-tolerant computing systems can be modeled as continuous-time Markov processes on large state spaces. Calculation of reliability is equivalent to computing transient probabilities of states of the Markov process corresponding to system failure. The state spaces are often very large, and thus efficient computational methods are required in order to calculate state probabilities. The CARE III approach has been developed to solve this problem; it is presented by Stiffler, Bryant, and Guccione [8] and further discussed by Trivedi and Geist [9]. The "randomization" technique is an alternate approach which is of considerable interest in its own right and which will be useful in validating the CARE III approach for systems with moderate state spaces.

The randomization modeling and computational technique will be illustrated on a simplified model of a fault-tolerant system consisting of three components similar to one presented by Trivedi and Geist [9]. Figure 1 shows the behavior of a single component. Initially the component

THE GEORGE WASHINGTON UNIVERSITY
School of Engineering and Applied Science
Department of Operations Research

Institute for Management Science and Engineering

Abstract
of
Serial T-469
29 October 1982

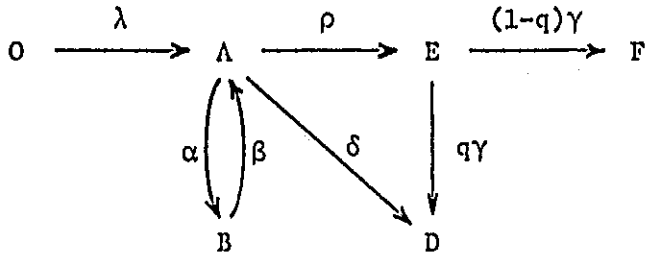
RELIABILITY CALCULATION USING RANDOMIZATION FOR MARKOVIAN FAULT-TOLERANT COMPUTING SYSTEMS

by

Douglas R. Miller

The randomization technique for computing transient probabilities of Markov processes is presented. The technique is applied to a Markov process model of a simplified fault-tolerant computer system for illustrative purposes. It is applicable to much larger and more complex models. Transient state probabilities are computed, from which reliabilities are derived. A new accelerated version of the randomization algorithm is developed which exploits "stiffness" of the models to gain increased efficiency. A great advantage of the randomizaton approach is that it easily allows probabilities and reliabilities to be computed to any predetermined accuracy.

Research Supported by
National Aeronautics and Space Administration



O : component OK

A : active fault

B : benign fault

D : detected (and reconfigured)

E : error

F : failure (error propagated)

Figure 1.--Single component reliability model: state space, transitions, and rates.

is fault-free, but after an exponential holding time with rate λ an "active" fault occurs. From the active state the fault may become "benign" and later become active and continue alternating between active and benign. From the active state the fault may be "detected" (by diagnostics) or generate an "error." This error may lead to detection of the fault and system reconfiguration or to "failure" of the system. Figure 1 shows the six states of a component, the possible transitions, and the rates at which they occur. We shall apply the randomization procedure to a system consisting of three independent components. The state space of the three-component system is shown in Figure 2; also shown are the possible transitions of the Markov process and their rates. The model has 18 states and 31 transitions. The set {F, AA, AE, BF, FD, XXA} is defined as "system failure," and the goal is to compute the

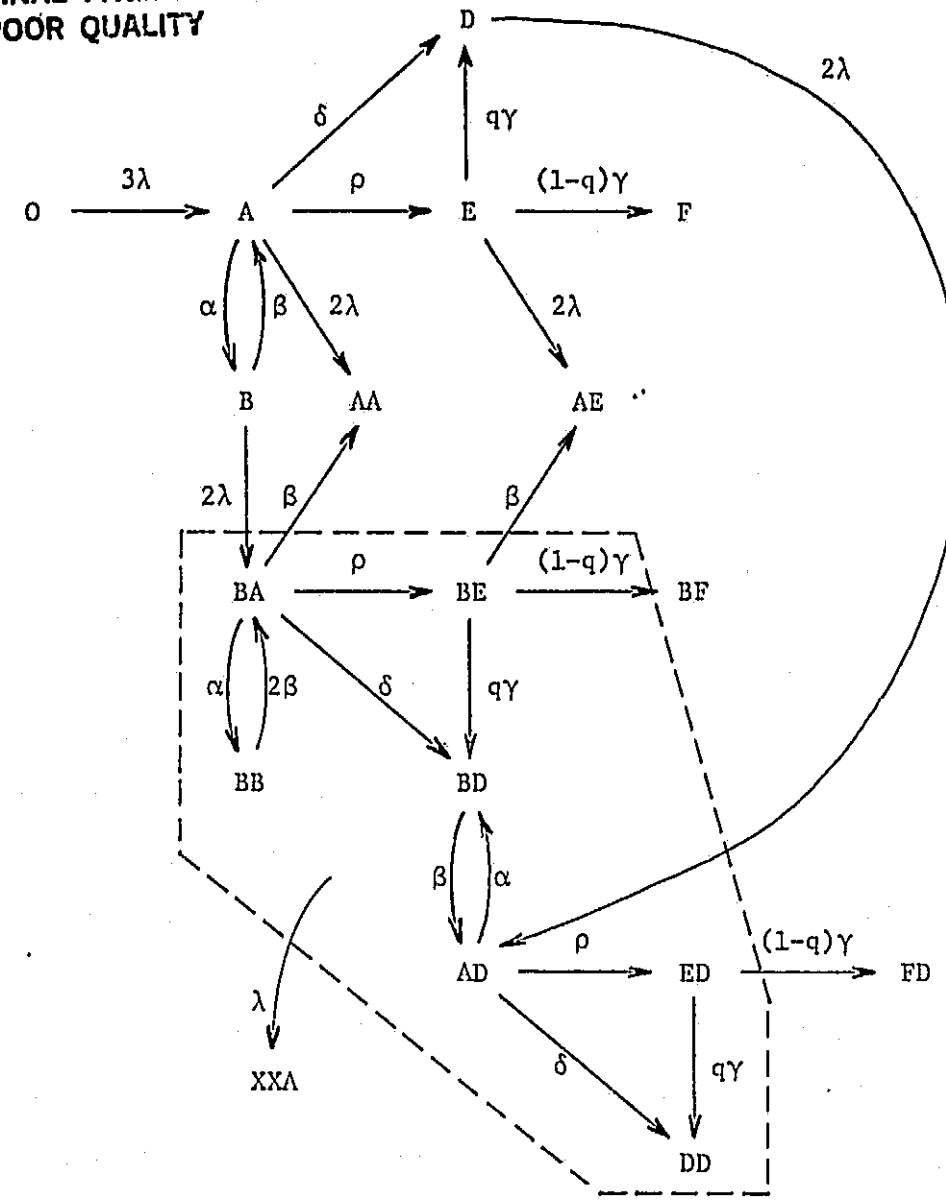


Figure 2.--A three-component system reliability model: state space, transitions, and rates.

probability of absorption of the process in this set at time t_M, the mission completion time. Note that this model is similar to one presented by Trivedi and Geist [9, p. 46]. It has a small state space, but it serves well as an illustrative example. The randomization technique

can be efficiently applied to models with much larger state spaces. It should be useful in analyses of two fault-tolerant computing systems developed under NASA sponsorship: SIFT [10] and FTMP [5].

The paper is structured as follows. Section 2 contains the standard randomization algorithm for computing transient probabilities of Markov processes. In Section 3 a new accelerated version of the randomization procedure is developed; it exploits the fact that models of many fault-tolerant computing systems are "stiff," i.e., the model has very fast and very slow transition rates. Section 4 gives computational results for the standard and accelerated algorithms applied to ten different versions of the three-component model of Figure 2. Section 5 contains summary comments and a brief discussion of other approaches. Two appendices contain listings of FORTRAN programs of the two algorithms. Additional information on the randomization technique may be found in Gross and Miller [3].

2. THE STANDARD RANDOMIZATION ALGORITHM

Let $\{X(t),\ t \geq 0\}$ be a continuous-time Markov process on a finite state space $S = \{1,2,\ldots,m\}$. The state probability vector at time t is denoted $\underline{\pi}(t) = (\pi_1(t), \pi_2(t), \ldots, \pi_m(t))$, where $\pi_s(t) = P(X(t) = s)$, $s \in S$. Two different characterizations of the stochastic nature of $\{X(t),\ t \geq 0\}$ are useful: (i) the infinitesimal generator and (ii) a randomized Markov chain.

All Markov processes can be characterized by an initial distribution $\underline{\pi}(0)$ and an infinitesimal generator

$$Q = \begin{pmatrix} -q_1 & q_{12} & q_{13} & \cdots & q_{1m} \\ q_{21} & -q_2 & q_{23} & \cdots & q_{2m} \\ q_{31} & q_{32} & -q_3 & \cdots & q_{3m} \\ \vdots & \vdots & \vdots & & \vdots \\ q_{m1} & q_{m2} & q_{m3} & \cdots & -q_m \end{pmatrix}$$

where

$$q_{ij} = \lim_{\Delta t \to 0} \frac{P(X(t+\Delta t)=y \mid X(t)=i)}{\Delta t} , \qquad i \neq j$$

and

$$q_i = \sum_{j \neq i} q_{ij} .$$

The q_{ij}'s are the transition rates which are depicted in Figures 1 and 2. The infinitesimal generator Q seems to be the most natural way to describe the stochastic nature of the Markov models of fault-tolerant computing systems.

Any Markov process on a finite state space can be represented as a discrete time Markov chain "randomized" by a Poisson process. Define

$$\Lambda = \max_{i \in S} q_i \tag{2.1}$$

and

$$P = Q/\Lambda + I , \tag{2.2}$$

where I is the identity matrix; P is a stochastic matrix. Let $\{Y_n, n = 0,1,2,\ldots\}$ be a Markov chain on S with transition matrix P and initial distribution $\underline{\pi}(0)$. Let $\{N(t), t \geq 0\}$ be a Poisson process with rate Λ which is independent of $\{Y_n, n = 0,1,2,\ldots\}$. Then $\{Y_{N(t)}, t \geq 0\}$ is a Markov process with generator Q and initial distribution $\underline{\pi}(0)$ and hence is probabilistically identical to $\{X(t), t \geq 0\}$.

(The relationship between sample paths of $\{Y_n,\ n = 0,1,2,\ldots\}$, $\{N(t),\ t \geq 0\}$, and $\{Y_{N(t)},\ t \geq 0\}$ is shown in Figure 3.) This construction makes it possible to compute transient probabilities of a Markov process with generator Q from transient probabilities of a Markov chain Y with transition matrix P and a Poisson process N with rate Λ. The transient probabilities of Y are denoted $\underline{\phi}(n) = (\phi_1(n), \phi_2(n), \ldots, \phi_m(n))$, where $\phi_s(n) = P(Y_n = s)$, $s \in S$. The randomization formula is

$$P(X(t) = s) = \sum_{n=0}^{\infty} P(X(t) = s \mid N(t) = n)\, P(N(t) = n)$$

$$= \sum_{n=0}^{\infty} P(Y_n = s)\, P(N(t) = n)$$

or equivalently,

$$\underline{\pi}(t) = \sum_{n=0}^{\infty} \underline{\phi}(n)\, \frac{e^{-\Lambda t}(\Lambda t)^n}{n!}. \tag{2.3}$$

See Gross and Miller [3] for additional discussion and details. (Equation (2.3) can also be found in Çinlar [1, p. 259].)

The infinite series in Equation (2.3) must be truncated for computational purposes. Let

$$T(\varepsilon,t) = \min\left[k:\ P(N(t) > k) \leq \varepsilon\right] = \min\left[k:\ \sum_{n=0}^{k} \frac{e^{-\Lambda t}(\Lambda t)^n}{n!} > 1 - \varepsilon\right] \tag{2.4}$$

where ε equals the acceptable error (specified by the user). The computational version of Equation (2.3) is

$$\underline{\pi}^{\varepsilon}(t) = \sum_{n=0}^{T(\varepsilon,t)} \underline{\phi}(n)\, \frac{e^{-\Lambda t}(\Lambda t)^n}{n!} \tag{2.5}$$

Truncation of the infinite series involves a probability loss of at most

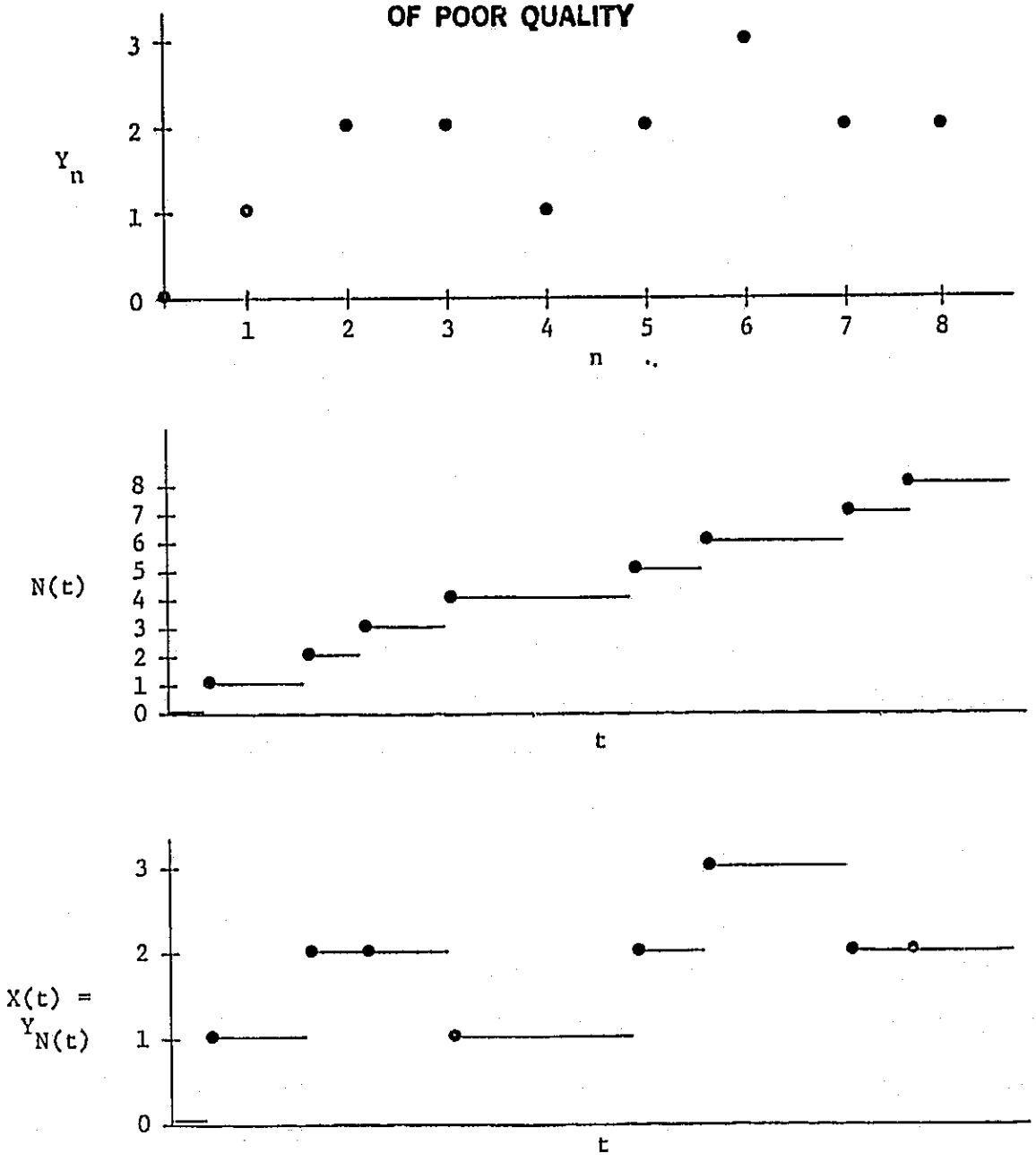


Figure 3.--Example of randomization: Markov chain Y_n, Poisson process N(t), and randomized chain X(t).

ε , thus all probabilities (of states or subsets of states) will have an error between $-\varepsilon$ and 0. Note that the randomization formula (2.5) reduces the calculation of transient probabilities of a Markov process

to those of a Markov chain and underlying Poisson process, both of which are more amenable to exact numerical evaluation.

The ϕ's are computed recursively using the relation from standard Markov chain theory:

$$\begin{aligned} \underset{\sim}{\phi}(0) &= \underset{\sim}{\pi}(0) \\ \underset{\sim}{\phi}(n+1) &= \underset{\sim}{\phi}(n)P \ , \qquad n \geqslant 0 \ . \end{aligned} \tag{2.6}$$

(Note that Equation (2.6) involves only nonnegative numbers, a fact that contributes to numerical stability of the algorithm.) The matrix P is usually sparse and thus the above matrix multiplication should be performed by an appropriate algorithm. Such a multiplication algorithm is described by Gross and Miller [3]. The number of operations in this algorithm is proportional to the sum of the number of states and the number of transitions, e.g., 49 for the system of Figure 2. The programs in the appendices use this multiplication algorithm.

In short, the standard randomization computational algorithm computes Λ and P from the generator Q using (2.1) and (2.2), respectively. It computes the truncation point $T(\varepsilon,t)$ from (2.4), then the $\underset{\sim}{\phi}(n)$'s using (2.6) recursively, accumulating in Equation (2.5) to give $\underset{\sim}{\pi}^{\varepsilon}(t)$. This algorithm was applied to ten versions of the model in Figure 2. The results are summarized in Section 4.

3. AN ACCELERATED ALGORITHM USING SELECTIVE RANDOMIZATION

A close investigation of the standard randomization algorithm and the model of the three-component system in Figure 2 suggests a way to speed up the algorithm for this kind of model. In the three-component model states 0 , D , and DD have very long mean holding times because

the component failure rate λ is very small. All other nonabsorbing states have much shorter holding times. The absorbing states have infinite holding times. The process $\{X(t),\ t \geq 0\}$ spends most of its time in the states with long holding times. The Markov chain $\{Y_n,\ n = 0,1,\ldots\}$ tends to sit in these states for many occurrences of the underlying Poisson process $\{N(t),\ t \geq 0\}$, making a null transition at each occurrence. By eliminating the computation involved in these null transitions for the states with the longest holding times, the speed of the algorithm can be increased.

Consider a modification of the model for the three-component system of Figure 2. The states AD, ED, ED, DD, FD, and DDA are each split into two states in order to distinguish whether or not the first fault is detected before the second fault occurs. The modified model is shown in Figure 4; it has 26 states and 42 transitions. This modification reveals (in Figure 5) a special block tree structure which can be exploited in an accelerated randomization algorithm. The structure consists of the process alternating between states with long holding times (S_1, S_3, S_5, and S_9 in Figure 5) and short holding times (S_2 and S_4), not returning to any subset after leaving it, and finally being absorbed into a terminal set of states (S_6, S_7, S_8, S_{10}). Larger, more realistic models of many fault-tolerant systems will tend to have this same structure. Such a model is depicted in Figure 6. States with no undetected faults will have long holding times while those with undetected faults will be short. (Systems that contain processes with significantly different time scales are called "stiff" in the literature on differential equations.)

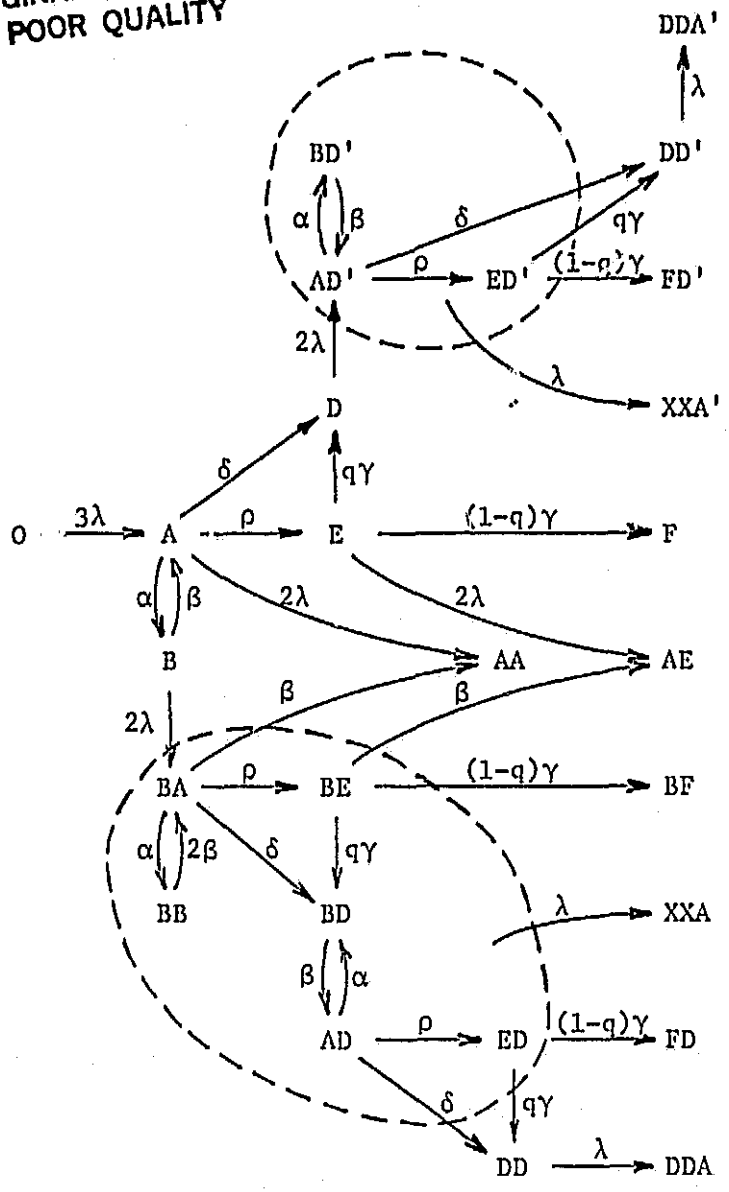


Figure 4.--Three-component system reliability model: modified state space with tree structure.

The accelerated randomization algorithm is based on a semi-Markov process representation of the Markov process $\{X(t), t \geq 0\}$. (Ross [7] presents Markov processes as a special case of semi-Markov processes.) This representation is also an extension of an idea called "selective randomization" by Melamed and Yadin [6]. Selective randomization is

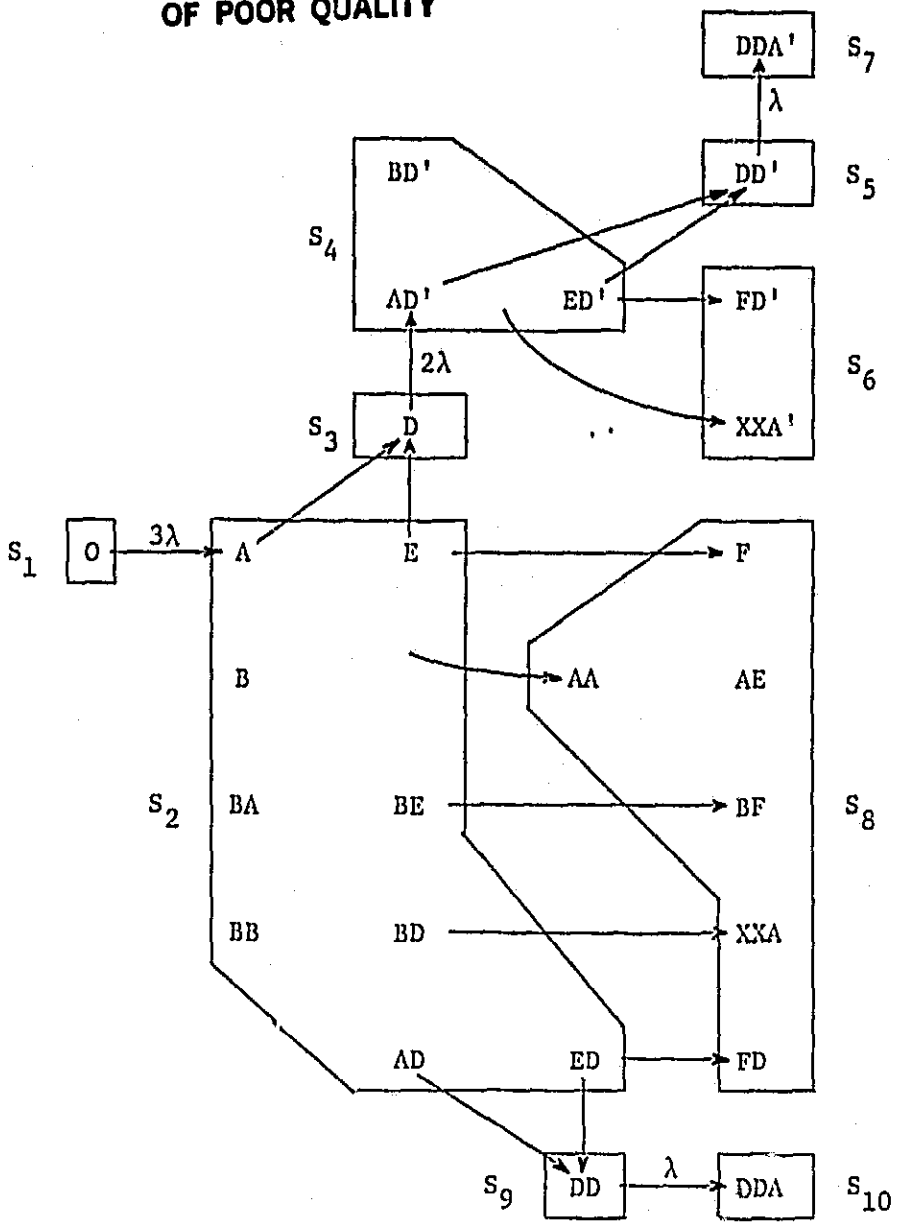


Figure 5.--Block structure of modified state space of three-component system reliability model.

similar to randomization except that the Markov chain is randomized by a Poisson process only while it is in a subset S_R of the state space S. The Markov chain is given arbitrary exponential holding times for the set $S^* = S - S_R$ of exceptional states. (In the model of Figures 4 and 5, $S_R = S_2 \cup S_4$.) Let

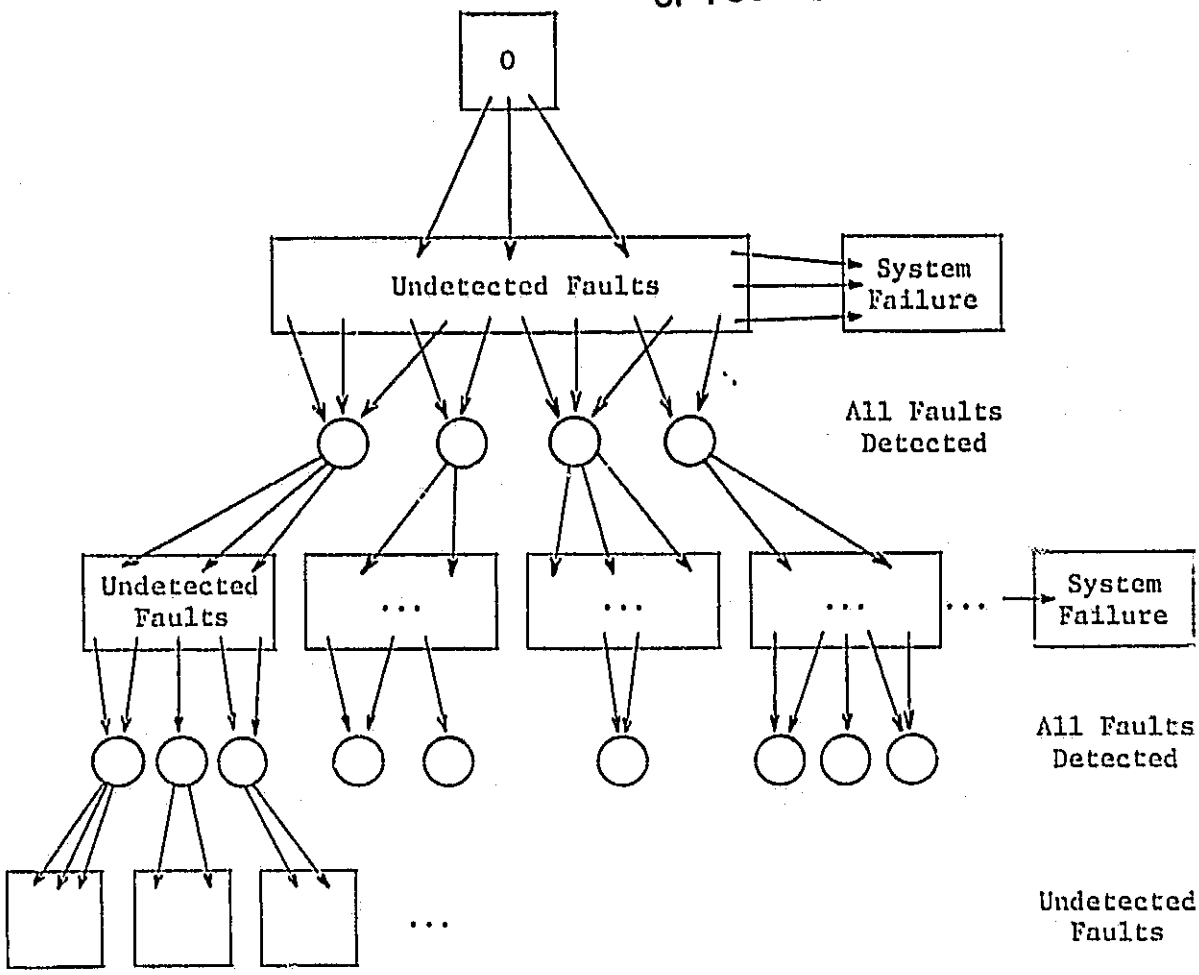


Figure 6.--Tree-like state space and transitions for a general model of fault-tolerant computers. (The initial state and states with all faults detected have very long mean holding times. States with undetected faults have very short mean holding times. System failure states have infinite holding times.)

$$\Lambda = \max_{s \in S_R} q_s = \max_{s \in S} q_s \,, \tag{3.1}$$

and define a substochastic transition matrix P^* ,

$$p^*_{ij} = \begin{cases} \dfrac{q_{i,j}}{\Lambda} + \delta_{i,j} & i \in S_R \\ \dfrac{q_{i,j}}{q_i} + \delta_{i,j} & q_i > 0,\ i \in S^* \\ 0 & q_i = 0,\ i \in S^* \,. \end{cases} \tag{3.2}$$

Then $\{X(t),\ t \geq 0\}$ can be characterized as a semi-Markov process with transition matrix P^* and exponential holding times with rates $\{r_i,\ i \in S\}$, where $r_i = \Lambda$ for $i \in S_R$ and $r_i = q_i$ for $i \in S^*$. (Thus the process is uniquely determined by $\underline{\pi}(0)$, P^*, Λ, and $\{q_i,\ i \in S^*\}$.) Let $\{Z_n,\ n = 0,1,\ldots\}$ be the embedded Markov chain with substochastic transition matrix P^*, and let $N^*(t)$ equal the number of transitions of Z in $[0,t]$, noting that Z may make transitions $i \to i$ for $i \in S_R$ but not for $i \in S^*$. Denote the transient probability vectors for Z by $\underline{\psi}(n)$, i.e., $\underline{\psi}(n) = (\psi_1(n), \ldots, \psi_m(n))$, where $\psi_s(n) = P(Z_n = s)$. The processes Z and N^* are dependent. The selective randomization formula is

$$\begin{aligned} P(X(t) = s) &= \sum_{n=0}^{\infty} P(X(t) = s,\ N^*(t) = n) \\ &= \sum_{n=0}^{\infty} P(Z_n = s,\ N^*(t) = n) \\ &= \sum_{n=0}^{\infty} P(Z_n = s)\ P(N^*(t) = n \mid Z_n = s) \\ &= \sum_{n=0}^{\infty} \psi_s(n)\ P(N^*(t) = n \mid Z_n = s) \end{aligned} \tag{3.3}$$

The accelerated randomization algorithm is based on Equation (3.3). The $\underline{\psi}$'s can be computed recursively,

$$\begin{aligned} \underline{\psi}(0) &= \underline{\pi}(0) \\ \underline{\psi}(n+1) &= \underline{\psi}(n)P^* \end{aligned} \tag{3.4}$$

In addition the quantities $P(N^*(t) = n \mid Z_n = s)$ must be computed, and the infinite series in Equation (3.3) must be truncated to achieve the desired numerical accuracy.

We note, for each subset S_i of S in Figure 5, that $P(N^*(t) = n \mid Z_n = s)$, $s \in S_i$, takes a constant value; for $i = 1,2,\ldots,10$,

$$A_{i,n}(t) = P(N^*(t) = n \mid Z_n = s) , \qquad s \in S_i ,$$

for n and s such that $P(Z_n = s) \neq 0$. Thus to use Equation (3.3) it suffices to compute $A_{i,n}(t)$, $i = 1,2,\ldots,10$. As an example, consider $A_{4,n}(t)$; this also equals the probability of $\{N^*(t) = n\}$ given $\{X(t) \in S_4\}$ occurs. A typical sample path depicting this situation is shown in Figure 7. In order to compute the probability of n occurrences in $[0,t]$ we revert to the standard randomization construction: the holding time in state 0 has an exponential distribution with rate 3λ . In the standard randomization, $p_{0,0} = 1 - (3\lambda/\Lambda) = (\Lambda - 3\lambda)/\Lambda$ and $p_{0,A} = 3\lambda/\Lambda$, and the transition to A will occur on the $(i + 1)$st occurrence of the underlying Poisson process with probability $(3\lambda/\Lambda)((\Lambda - 3\lambda)/\Lambda)^i$, $i = 0,1,2,\ldots$. Similarly the holding time in state D has an exponential distribution with rate 2λ and the process will leave D after being there for exactly j occurrences of the underlying Poisson process with probability $(2\lambda/\Lambda)((\Lambda - 2\lambda)/\Lambda)^j$, $j = 0,1,2,\ldots$. Consequently there are many ways that $\{N^*(t) = n\}$ can occur, depending on the number of occurrences of $\{N(t),\ t \geq 0\}$ that happen while $\{X(t),\ t \geq 0\}$ is holding in 0 or D. Combining all these facts gives

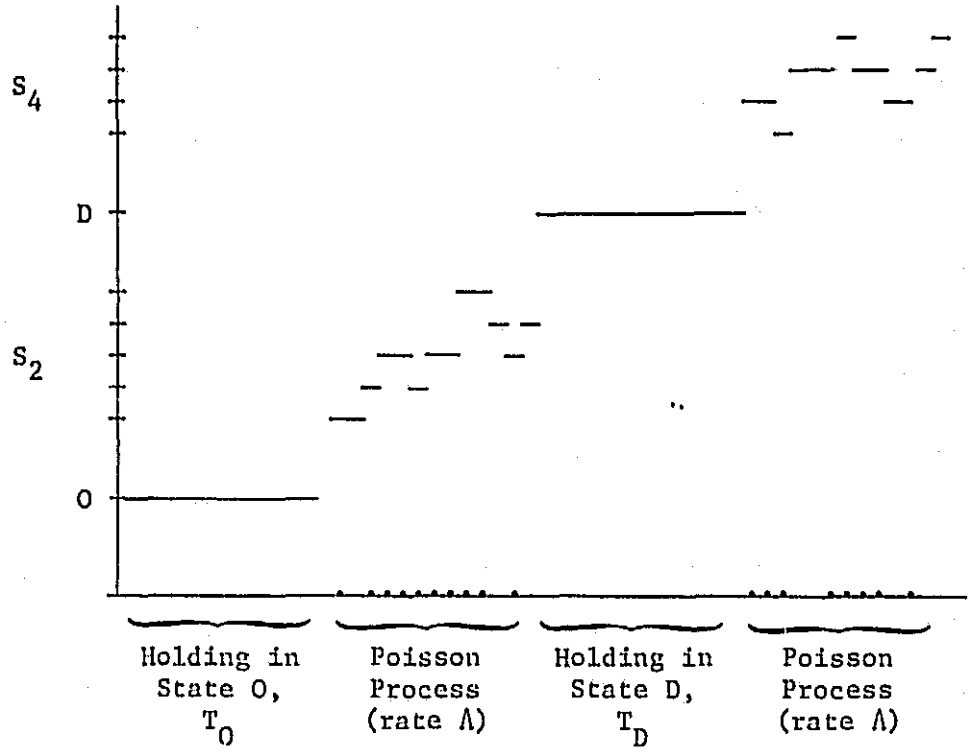


Figure 7.--A typical sample path of process on modified state space for three-component system.

$$A_{4,n}(t) = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \frac{3\lambda}{\Lambda} \left(\frac{\Lambda-3\lambda}{\Lambda}\right)^i \frac{2\lambda}{\Lambda} \left(\frac{\Lambda-2\lambda}{\Lambda}\right)^j P\{N(t) = n+i+j\}$$

$$= \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} \frac{3\lambda}{\Lambda} \left(\frac{\Lambda-3\lambda}{\Lambda}\right)^i \frac{2\lambda}{\Lambda} \left(\frac{\Lambda-2\lambda}{\Lambda}\right)^j \frac{e^{-\Lambda t}(\Lambda t)^{n+i+j}}{(n+i+j)!}$$

$$= \frac{6\lambda}{\Lambda} \left\{ \left(\frac{\Lambda}{\Lambda-2\lambda}\right)^{n-1} \left[e^{-2\lambda t} - e^{-\Lambda t} \sum_{k=0}^{n-1} \frac{[(\Lambda-2\lambda)t]^k}{k!} \right] \right.$$

$$\left. - \left(\frac{\Lambda}{\Lambda-3\lambda}\right)^{n-1} \left[e^{-3\lambda t} - e^{-\Lambda t} \sum_{k=0}^{n-1} \frac{[(\Lambda-3\lambda)t]^k}{k!} \right] \right\} .$$

Then analysis of $A_{i,n}(t)$ for other sets is based on the same principle. These quantities are:

$$
\begin{aligned}
A_{1,n}(t) &= e^{-3\lambda} , \qquad n = 0 , \\
A_{2,n}(t) &= \frac{3\lambda}{\Lambda} a_3^n f_3(n) , \qquad n \geq 1 , \\
A_{3,n}(t) &= 3a_2^{n-1} f_2(n) - 3a_3^{n-1} f_3(n) , \qquad n \geq 2 , \\
A_{4,n}(t) &= \frac{2\lambda}{\Lambda} A_{3,n}(t) , \qquad n \geq 2 , \\
A_{5,n}(t) &= 3a_1^{n-2} f_1(n) - 6a_2^{n-2} f_2(n) + 3a_3^{n-2} f_3(n) , \qquad n \geq 3 , \\
A_{6,n}(t) &= f_0(n) - 3a_2^{n-2} f_2(n) + 2a_3^{n-2} f_3(n) , \qquad n \geq 3 , \\
A_{7,n}(t) &= f_0(n) - 3a_1^{n-3} f_1(n) + 3a_2^{n-3} f_2(n) - a_3^{n-3} f_3(n) , \qquad n \geq 4 , \\
A_{8,n}(t) &= f_0(n) - a_3^{n-1} f_3(n) , \qquad n \geq 2 , \\
A_{9,n}(t) &= \frac{3}{2} a_1^{n-1} f_1(n) - \frac{3}{2} a_3^{n-1} f_3(n) , \qquad n \geq 2 , \\
A_{10,n}(t) &= f_0(n) - \frac{3}{2} a_1^{n-2} f_1(n) + \frac{1}{2} a_3^{n-2} f_3(n) , \qquad n \geq 3 ,
\end{aligned}
\tag{3.5}
$$

where, for $i = 0,1,2,3$,

$$f_i(n) = e^{-i\lambda t} - \sum_{k=0}^{n-1} e^{-\lambda t} \frac{[(\Lambda - i\lambda)t]^k}{k!}$$

$$a_i = \frac{\Lambda}{\Lambda - i\lambda} .$$

These equations provide the required probabilities for Equation (3.3).

To complete the specification of the accelerated randomization algorithm it is necessary to give a truncation rule for the infinite series in Equation (3.3). Note that

$$P(N^*(t) \leq \ell) = \sum_{n=0}^{\ell} \sum_{i=1}^{10} P(X(t) \in S_i,\ N^*(t) = n)$$

$$= \sum_{n=0}^{\ell} \sum_{i=1}^{10} P(Z_n \in S_i)\ A_{i,n}(t) \qquad (3.6)$$

$$= \sum_{n=0}^{\ell} \sum_{i=1}^{10} A_{i,n}(t) \sum_{s \in S_i} \psi_s(n)$$

and define

$$T^*(\varepsilon,t) = \min\left\{\ell:\ P(N^*(t) \leq \ell) \geq 1 - \varepsilon\right\} . \qquad (3.7)$$

Thus, to summarize the accelerated randomization algorithm: the $\underline{\psi}$'s are computed recursively using Equation (3.4) and the A's are computed using (3.5) with the products being accumulated in Equations (3.3) and (3.6) until the truncation point T^* of Equation (3.7) is met, at which point the algorithm terminates, yielding probabilities which are accurate to within ε of the exact values.

The accelerated randomization algorithm was programmed (see the appendix for FORTRAN listing) for the model of Figure 4 and executed for the same ten versions of the system as the standard algorithm. Results are summarized in Section 4.

4. COMPUTATIONAL RESULTS

The standard randomization algorithm and the accelerated randomization algorithm have been programmed in FORTRAN for the system depicted in Figures 2 and 4, respectively. Listings of the programs appear in the appendix. Ten cases were run with different sets of parameter values. The input values for the different cases are given in Table I. (Note that cases 3 and 4 and cases 5 and 6 are identical except for the user

TABLE I

CASES COMPUTED

Case	Parameters								
	λ	α	β	δ	γ	ρ	q	t_M	ε
1	10^{-3}	10	1	10	100	100	.99	1	10^{-9}
2	10^{-3}	10	1	10	100	100	.99	10	10^{-9}
3	10^{-3}	10	10	10	100	100	.99	1	10^{-9}
4	10^{-3}	10	10	10	100	100	.99	1	10^{-7}
5	10^{-3}	10	10	10	100	100	.99	10	10^{-9}
6	10^{-3}	10	10	10	100	100	.99	10	10^{-7}
7	10^{-3}	100	100	100	10^4	10^4	.99	1	10^{-9}
8	10^{-3}	100	100	100	10^4	10^4	.99	10	10^{-9}
9	10^{-3}	10^4	10^4	10^4	10^4	10^4	.99	1	10^{-9}
10	10^{-4}	100	100	100	10^4	10^4	.99	1	10^{-9}

specified error bound, ε .) The programs were run on The George Washington University's IBM 370/4341.

The execution times of these randomization programs are proportional to the product of the truncation points (T or T*) and the sum of the number of states plus the number of transitions. The accelerated version requires more CPU time for each term in the randomization formula (3.3) because the weights $A_{i,n}(t)$, i = 1,2,...,10 require more computation time. However, for systems with large state spaces this will be insignificant compared to the calculation in Equation (3.4). Thus performance is more accurately predicted by the number of terms

multiplied by $18 + 31 = 49$ for the standard algorithm and $26 + 42 = 68$ for the accelerated algorithm. The number of terms (truncation point) and CPU times in seconds are summarized in Table II. In most cases the accelerated algorithm appears to be far superior.

The actual probabilities computed are presented in Table III. The probabilities listed are for the accelerated modification. The probabilities from the standard algorithm agree completely with these numbers and may be recovered from Table III by summing the probabilities for the split states, e.g., $P(AD) + P(AD')$.

TABLE II

PERFORMANCE OF RANDOMIZATION ALGORITHMS:
NUMBER OF TERMS REQUIRED AND CPU TIME

Case	Standard Algorithm		Accelerated Algorithm	
	No. Terms	CPU Seconds	No. Terms	CPU Seconds
1	194	3.20	159	6.50
2	1424	19.54	1233	48.81
3	204	3.12	130	5.40
4	193	3.06	95	4.05
5	1522	20.02	206	8.39
6	1492	20.23	143	5.79
7	1522	20.01	174	7.16
8	13656	174.51	208	8.40
9	4383	56.43	95	4.03
10	1522	19.59	143	5.86

TABLE III

STATE PROBABILITIES COMPUTED

State	Probabilities							
	Case 1	Case 2	Case 3	Case 5	Case 7	Case 8	Case 9	Case 10
O	.997004496	.970445534	.997004496	.970445534	.997004496	.970445534	.997004496	.999700045
A	.000026271	.000026469	.000027191	.000026467	.000002719	.000002647	.000001496	.000000273
B	.000162350	.000264929	.000027190	.000026470	.000002719	.000002647	.000001496	.000000273
E	.000026263	.000026469	.000027191	.000026467	.000002719	.000002647	.000001496	.000000273
BA	.000000003	.000000005	.000000000	.000000000	.000000000	.000000000	.000000000	.000000000
BB	.000000009	.000000024	.000000000	.000000000	.000000000	.000000000	.000000000	.000000000
BE	.000000003	.000000005	.000000000	.000000000	.000000000	.000000000	.000000000	.000000000
BD	.000000124	.000000563	.000000005	.000000005	.000000000	.000000000	.000000000	.000000000
AD	.000000001	.000000005	.000000000	.000000000	.000000000	.000000000	.000000000	.000000000
ED	.000000001	.000000005	.000000000	.000000000	.000000000	.000000000	.000000000	.000000000
D	.002752397	.028678470	.002884497	.028914765	.002957223	.028985558	.002973089	.000296387
AD'	.000000047	.000000516	.000000051	.000000525	.000000005	.000000053	.000000003	.000000000
BD'	.000000167	.000004641	.000000046	.000000520	.000000005	.000000053	.000000003	.000000000
ED'	.000000046	.000000516	.000000051	.000000524	.000000005	.000000053	.000000003	.000000000

Table III--*continued*

State	Probabilities							
	Case 1	Case 2	Case 3	Case 5	Case 7	Case 8	Case 9	Case 10
DD'	.000002401	.000277408	.000002641	.000285620	.000002908	.000288426	.000002946	.000000029
FD'	.000000022	.000002553	.000000024	.000002629	.000000027	.000002655	.000000015	.000000000
XXA'	.000000000	.000000026	.000000000	.000000008	.000000000	.000000001	.000000000	.000000000
DDA'	.000000001	.000000915	.000000001	.000000952	.000000001	.000000966	.000000001	.000000000
F	.000025252	.000265710	.000026464	.000267916	.000027155	.000268601	.000014948	.000002719
AA	.000000053	.000000575	.000000057	.000000581	.000000006	.000000058	.000000004	.000000000
AE	.000000052	.000000574	.000000056	.000000576	.000000006	.000000058	.000000003	.000000000
BF	.000000002	.000000042	.000000000	.000000004	.000000000	.000000000	.000000000	.000000000
FD	.000000000	.000000037	.000000000	.000000004	.000000000	.000000000	.000000000	.000000000
XXA	.000000000	.000000005	.000000000	.000000000	.000000000	.000000000	.000000000	.000000000
DD	.000000040	.000003989	.000000033	.000000430	.000000004	.000000044	.000000001	.000000000
DDA	.000000000	.000000016	.000000000	.000000002	.000000000	.000000000	.000000000	.000000000

5. CONCLUSIONS

Randomization appears to be a good way to compute reliabilities for Markovian fault-tolerant computing systems with state spaces of moderate size. Gross and Miller [4] have solved Markov processes with 20,000 states and 200,000 transitions using the standard randomization procedure. It is certainly feasible to use the approach on Markovian models of fault-tolerant systems of comparable or even larger size.

The accelerated randomization algorithm gives a significant savings in CPU time for most examples. There should be an even greater savings for larger systems. Furthermore, this accelerated implementation is applicable to any passage time problem, the exceptional set S^* being the target states (with holding times set to infinity). This has application in computing fault-recovery-time distributions for fault-tolerant systems.

The randomization algorithm is quite easy to implement. The main difficulty encountered in larger systems would be generation of the Q matrix. It is necessary to have an automatic way for the computer to generate Q or a sparse representation of it. Fortunately, the SERT methodology (see Gross and Miller [3]) can be applied to models of fault-tolerant systems to overcome this difficulty.

The usual approach to computing transient probabilities for Markov processes is solution of the Kolmogorov forward equation

$$\underset{\sim}{\pi}'(t) = \underset{\sim}{\pi}(t)Q \ , \qquad t \geqslant 0 \ .$$

This is an initial value system with $\underset{\sim}{\pi}(0)$ given. There are two general approaches: (i) numerical integration techniques such as Runge-Kutta, predictor-corrector, etc., and (ii) exponentiation [$\underset{\sim}{\pi}(t) = \underset{\sim}{\pi}(0)e^{Qt}$] by computing the spectrum, computing the Taylor series, or other means.

The randomization technique has a distinct advantage over these approaches in that a bound on the global error can be set by the user, and it is achieved with certainty. Furthermore, Grassmann [2] has shown randomization to be more efficient for some queuing systems.

Another advantage of the randomization approach is that it is a "computational probability" technique. Computational probability is an emerging discipline concerned with numerical solution of applied probability problems. The probabilistic structure of the model is exploited to obtain efficient numerical algorithms and to evaluate the performance of algorithms. In this particular application, probabilistic reasoning led to the accelerated algorithm. Another benefit of the probabilistic analysis is that Equation (2.3) just involves nonnegative numbers, creating numerical stability. Finally, the probabilistic point of view leads to efficient numerical algorithms for computing other quantities of interest, for example, occupancy time distributions and expectations which can be used in a performability analysis.

Over all, it appears that the randomization technique is a very promising methodology for calculating reliabilities and related quantities for Markovian fault-tolerant computing systems.

ACKNOWLEDGMENT

Mr. Leonidas Kioussis wrote the FORTRAN programs implementing the randomization algorithms in this paper. His assistance is greatly appreciated.

REFERENCES

[1] Çinlar, E. (1975). *Introduction to Stochastic Processes*. Prentice-Hall, Englewood Cliffs, New Jersey.

[2] Grassmann, W. (1977). Transient solutions in Markovian queueing systems. *Computers & Operations Research*, Vol. 4, pp. 47-56.

[3] Gross, D. and D. R. Miller (1982). The randomization technique as a modelling tool and solution procedure for transient Markov processes. Technical Paper Serial T-467, Institute for Management Science and Engineering, The George Washington University, August.

[4] Gross, D. and D. R. Miller (1982). Multi-echelon repairable item provisioning in a time varying environment using the randomization technique. Technical Paper Serial T-468, Institute for Management Science and Engineering, The George Washington University, September.

[5] Hopkins, A. H. Jr., T. B. Smith III, and J. H. Lala (1978). FTMP--A highly reliable fault-tolerant multiprocessor for aircraft control. *Proceedings IEEE*, Vol. 66, pp. 1221-1239.

[6] Melamed, B. and M. Yadin (1980). Randomization procedures in the computation of cumulative time distributions over discrete state Markov processes. Preprint.

[7] Ross, S. M. (1970). *Applied Probability Models with Optimization Applications*. Holden-Day, San Francisco.

[8] Stiffler, J. J., L. A. Bryant, and L. Guccione (1979). CARE III final report, phase I. NASA Contractor Report 159122, NASA Langley Research Center, Hampton, Virginia.

[9] Trivedi, K. S. and R. M. Geist (1981). A tutorial on the CARE III approach to reliability modeling. NASA Contractor Report 3488, NASA Langley Research Center, Hampton, Virginia.

[10] Wensley, J. H., L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, M. P. Melliar-Smith, R. E. Shostak, and C. B. Weinstock (1978). SIFT: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings IEEE*, Vol. 66, pp. 1240-1255.

APPENDIX 1

STANDARD RANDOMIZATION PROGRAM

```
C          THIS PROGRAM IS SPECIALIZED FOR A PARTICULAR
C          MARKOV PROCESS WITH :
C          18  STATES , 32  TRANSITIONS ,  6  ABSORBING  STATES  ,
C          AND ONE INITIAL STATE ( E.G. STATE NO.1 )
C          THE ABSORBING STATES ARE : STATES NO. 6,7,8,13, 17,18
C          ------------------------------------------------------------
C
           INTEGER TSTAR(50)
           DOUBLE PRECISION RSTAR(50),PSTAR(50)
           DOUBLE PRECISION PHIOLD(25),PHINEW(25),PRO(25),RLOPRO(25)
           DOUBLE PRECISION RLAMDA,ALPHA,BETA,DELTA,RHO,QUE,EPSILO
           DOUBLE PRECISION ERROR,T0,LT,TERM,TRSUM,PRLT,RLAMAX,RK,COUN
C
C-------------------------------------------------------------------------
C
C          READIN INPUT PARAMETERS
C
           READ, RLAMDA,ALPHA,BETA,DELTA,RHO,QUE,EPSILO
           PRINT, ' LAMDA : ',RLAMDA
           PRINT, ' ALPHA : ',ALPHA
           PRINT, ' BETA  : ',BETA
           PRINT, ' DELTA : ',DELTA
           PRINT, ' RHO   : ',RHO
           PRINT, ' QUE   : ',QUE
           PRINT, ' EPSILON:',EPSILO
           READ, T0
           READ, ERROR
           PRINT, ' ERROR : ',ERROR
           READ(5,7)    (TSTAR(I),I=1,50)
   7       FORMAT(15I4/15I4/15I4/5I4)
           NN = 18
C
C          CONSTRUCT THE RSTAR VECTOR
C
           RSTAR(1) = - 3 * RLAMDA
           RSTAR(2) = 3 * RLAMDA
           RSTAR(3) = - ( ALPHA + RHO + DELTA + 2 * RLAMDA )
           RSTAR(4) = ALPHA
           RSTAR(5) = RHO
           RSTAR(6) = DELTA
           RSTAR(7) = 2 * RLAMDA
           RSTAR(8) = - ( BETA + 2 * RLAMDA )
           RSTAR(9) = BETA
           RSTAR(10) = 2 * RLAMDA
           RSTAR(11) = - ( EPSILO + 2 * RLAMDA )
           RSTAR(12) = QUE * EPSILO
           RSTAR(13) = ( 1.0 - QUE ) * EPSILO
           RSTAR(14) = 2 * RLAMDA
           RSTAR(15) = - 2 * RLAMDA
           RSTAR(16) = 2 * RLAMDA
           RSTAR(17) = 0.0
           RSTAR(18) = 0.0
           RSTAR(19) = 0.0
```

```
      RSTAR(20) = - ( BETA + ALPHA + RHO + DELTA + RLAMDA )
      RSTAR(21) = BETA
      RSTAR(22) = ALPHA
      RSTAR(23) = RHO
      RSTAR(24) = DELTA
      RSTAR(25) = RLAMDA
      RSTAR(26) = - ( 2 * BETA + RLAMDA )
      RSTAR(27) = 2 * BETA
      RSTAR(28) = RLAMDA
      RSTAR(29) = - ( BETA + EPSILO + RLAMDA )
      RSTAR(30) = BETA
      RSTAR(31) = QUE * EPSILO
      RSTAR(32) = ( 1.0 - QUE ) * EPSILO
      RSTAR(33) = RLAMDA
      RSTAR(34) = - ( BETA + RLAMDA )
      RSTAR(35) = BETA
      RSTAR(36) = RLAMDA
      RSTAR(37) = 0.0
      RSTAR(38) = - ( ALPHA + RHO + DELTA + RLAMDA )
      RSTAR(39) = ALPHA
      RSTAR(40) = RHO
      RSTAR(41) = DELTA
      RSTAR(42) = RLAMDA
      RSTAR(43) = - ( EPSILO + RLAMDA )
      RSTAR(44) = QUE * EPSILO
      RSTAR(45) = (1.0 - QUE ) * EPSILO
      RSTAR(46) = RLAMDA
      RSTAR(47) = - RLAMDA
      RSTAR(48) = RLAMDA
      RSTAR(49) = 0.0
      RSTAR(50) = 0.0
C
C
      DO 18 I=1,NN
18    PHIOLD(I) = 0.0
      PHIOLD(1) = 1.0
C
C     COMPUTE  RLAMAX
C
      RLAMAX = RSTAR(1)
      DO 19 I=2,50
      IF ( RSTAR(I) .LE. RLAMAX ) RLAMAX = RSTAR(I)
19    CONTINUE
      RLAMAX = - RLAMAX
      PRINT, ' RLAMAX : ',RLAMAX
C
      COUN = DLOG( RLAMAX * T0 )
C
```

```
C          CONSTRUCTION OF THE PSTAR VECTOR
C
      DO 20 I=1,50
   20 PSTAR(I) = RSTAR(I)/RLAMAX
      DO 21 I=1,50
      IF ( PSTAR(I) .GT. 0.0 ) GO TO 21
      PSTAR(I) = 1.0 + PSTAR(I)
   21 CONTINUE
C
C.....
C
      K = 0
      LT = - RLAMAX * T0
      IF ( LT .GT. -120 ) THEN DO
      TERM = DEXP( LT )
      ELSE DO
      TERM = 0.0
      END IF
      TRSUM = TERM
      DO 27 I=1,NN
   27 PRO(I) = PHIOLD(I) * TERM
      IF ( TRSUM .GT. ( 1.0 - ERROR ) ) GO TO 100
   29 CONTINUE
      PRLT = LT
      K = K + 1
      RK = K
      CALL TRUNC(COUN,RK,PRLT,LT,TERM)
      CALL EVAL(NN,PHIOLD,PSTAR,TSTAR,PHINEW)
      DO 33 I=1,NN
      PHIOLD(I) = PHINEW(I)
   33 PRO(I) = PRO(I) + PHINEW(I) * TERM
      TRSUM = TRSUM + TERM
      IF ( TRSUM .GE. ( 1.0 - ERROR ) ) GO TO 100
      GO TO 29
  100 CONTINUE
      DO 110 I=1,NN
      RLOPRO(I) = DLOG10(PRO(I))
  110 CONTINUE
      WRITE(6,130)
  130 FORMAT(' ',///' ',5X,' TSTAR ',20X,' RSTAR ',30X,' PSTAR ',//)
      DO 135 I=1,50
      WRITE(6,140) TSTAR(I),RSTAR(I),PSTAR(I)
  140 FORMAT(' ',7X,I4,7X,F28.23,6X,F28.23)
  135 CONTINUE
      WRITE(6,150) K
  150 FORMAT(' ',///' ',' NUMBER OF TERMS SUMMED :',I17)
      WRITE(6,160) T0
  160 FORMAT(' ',///' ',' TIME OF INTEREST :',F10.6)
      WRITE(6,162)
  162 FORMAT(' ',/////' ',4X,' THE LOGARITHM OF THE PROBABILITIES IS : '
      DO 165 I=1,NN
      WRITE(6,170) I,RLOPRO(I),PRO(I)
```

```
 170   FORMAT(' ///' ',5X,' LOGPRO(',I3,' ) = ',F30.26,5X,F30.26)
 165   CONTINUE
       STOP
       END
C
C.....................................................................
C
       SUBROUTINE EVAL(NN,PHIOLD,PSTAR,TSTAR,PHINEW)
       DOUBLE PRECISION PHIOLD(25),PSTAR(50),PHINEW(25)
       INTEGER TSTAR(50)
       DOUBLE PRECISION PHIJ
       DO 33 J=1,NN
       PHINEW(J) = 0.0
  33   CONTINUE
       I = 0
       DO 2 J=1,NN
       I = I + 1
       PHIJ = PHIOLD(J)
       PHINEW(J) = PHIJ * PSTAR(I) + PHINEW(J)
       MJ = TSTAR(I)
       IF ( MJ .EQ. 0 ) GO TO 2
       DO 1 K=1,MJ
       I = I + 1
       LJK = TSTAR(I)
   1   PHINEW(LJK) = PHINEW(LJK) + PHIJ * PSTAR(I)
   2   CONTINUE
       RETURN
       END
C
C.....................................................................
C
       SUBROUTINE TRUNC(COUN,RK,PRLT,LT,TERM)
       DOUBLE PRECISION COUN,RK,PRLT,LT,TERM
       LT = PRLT + COUN - DLOG( RK )
       IF ( LT .GT. -120 ) THEN DO
       TERM = DEXP( LT )
       ELSE DO
       TERM = 0.0
       END IF
       RETURN
       END
C
C.....................END OF PROGRAM LK.....................
```

APPENDIX 2

ACCELERATED RANDOMIZATION ALGORITHM

```
C             THIS PROGRAM IS SPECIALIZED FOR A PARTICULAR
C             MARKOV PROCESS WITH :
C             26 STATES , 42 TRANSITIONS , 10 ABSORBING STATES
C
      IMPLICIT REAL*8(A-H, O-Z), INTEGER(I-N)
      INTEGER TSTAR(68)
      DIMENSION RSTAR(68),PSTAR(68)
      DIMENSION PHIOLD(26),PHINEW(26),PRO(26),RLOPRO(26)
      DIMENSION PIK(10),PSI(10),DLA(4)
      DIMENSION COUN(4),ST(4),PRST(4),PRLT(4)
      DOUBLE PRECISION LT(4)
      DOUBLE PRECISION ALG(3),FI(4)
      DATA XL/1.00D-62/
C
C-----------------------------------------------------------------------
C
C             READIN  INPUT  PARAMETERS
C
      READ, RLAMDA,ALPHA,BETA,DELTA,RHO,QUE,EPSILO
      PRINT, ' LAMDA :',RLAMDA
      PRINT, ' ALPHA :',ALPHA
      PRINT, ' BETA  :',BETA
      PRINT, ' DELTA :',DELTA
      PRINT, ' RHO   :',RHO
      PRINT, ' QUE   :',QUE
      PRINT, ' EPSILON:',EPSILO
      READ, T0
      READ, ERROR
      PRINT, ' ERROR :',ERROR
      READ, NN
      READ(5,1) (TSTAR(I),I=1,68)
    1 FORMAT(3012/3012/8I2)
C
C            THE  DLA  PARAMETER VECTOR
C
      DLA(1) = 3 * RLAMDA
      DLA(2) = 2 * RLAMDA
      DLA(3) = RLAMDA
      DLA(4) = 0.0
C
      DXP1 = DEXP( - ( DLA(1) * T0))
      PRINT, ' DXP1 :',DXP1
      DXP2 = DEXP( - ( DLA(2) * T0))
      PRINT, ' DXP2 :',DXP2
      DXP3 = DEXP( - ( DLA(3) * T0))
      PRINT, ' DXP3 :',DXP3
C
C              CONSTRUCT  THE  RSTAR  VECTOR
C
         RSTAR(1) = 0.0
         RSTAR(2) = 0.0
         RSTAR(3) = - ( ALPHA + RHO + DELTA + 3 * RLAMDA )
         RSTAR(4) = ALPHA
```

```
      RSTAR(5) = RHO
      RSTAR(6) = DELTA
      RSTAR(7) = 2 * RLAMDA
      RSTAR(8) = - ( BETA + 2 * RLAMDA )
      RSTAR(9) = BETA
      RSTAR(10) = 2 * RLAMDA
      RSTAR(11) = - ( EPSILO + 2 * RLAMDA )
      RSTAR(12) = QUE * EPSILO
      RSTAR(13) = ( 1.0 - QUE ) * EPSILO
      RSTAR(14) = 2 * RLAMDA
      RSTAR(15) = - ( ALPHA + BETA + RHO + DELTA + RLAMDA )
      RSTAR(16) = ALPHA
      RSTAR(17) = RHO
      RSTAR(18) = DELTA
      RSTAR(19) = BETA
      RSTAR(20) = RLAMDA
      RSTAR(21) = - ( 2 * BETA + RLAMDA )
      RSTAR(22) = 2 * BETA
      RSTAR(23) = RLAMDA
      RSTAR(24) = - ( BETA + EPSILO + RLAMDA )
      RSTAR(25) = QUE * EPSILO
      RSTAR(26) = BETA
      RSTAR(27) = ( 1.0 - QUE ) * EPSILO
      RSTAR(28) = RLAMDA
      RSTAR(29) = - ( BETA + RLAMDA )
      RSTAR(30) = BETA
      RSTAR(31) = RLAMDA
      RSTAR(32) = - ( ALPHA + RHO + DELTA + RLAMDA )
      RSTAR(33) = ALPHA
      RSTAR(34) = RHO
      RSTAR(35) = RLAMDA
      RSTAR(36) = DELTA
      RSTAR(37) = - ( EPSILO + RLAMDA )
      RSTAR(38) = ( 1.0 - QUE ) * EPSILO
      RSTAR(39) = RLAMDA
      RSTAR(40) = QUE * EPSILO
      RSTAR(41) = 0.0
      RSTAR(42) = 0.0
      RSTAR(43) = - ( ALPHA + RHO + DELTA + RLAMDA )
      RSTAR(44) = ALPHA
      RSTAR(45) = RHO
      RSTAR(46) = DELTA
      RSTAR(47) = RLAMDA
      RSTAR(48) = - ( BETA + RLAMDA )
      RSTAR(49) = BETA
      RSTAR(50) = RLAMDA
      RSTAR(51) = - ( EPSILO + RLAMDA )
      RSTAR(52) = QUE * EPSILO
      RSTAR(53) = ( 1.0 - QUE ) * EPSILO
      RSTAR(54) = RLAMDA
      RSTAR(55) = 0.0
      RSTAR(56) = 0.0
```

```
      RSTAR(57) = 0.0
      RSTAR(58) = 0.0
      RSTAR(59) = 0.0
      RSTAR(60) = 0.0
      RSTAR(61) = 0.0
      RSTAR(62) = 0.0
      RSTAR(63) = 0.0
      RSTAR(64) = 0.0
      RSTAR(65) = 0.0
      RSTAR(66) = 0.0
      RSTAR(67) = 0.0
      RSTAR(68) = 0.0
C
C
      DO 5 I=1,26
      PHIOLD(I) = 0.0
 5    CONTINUE
      PHIOLD(1) = 1.0
C
C                    COMPUTE  RLAMAX
C
      RLAMAX = RSTAR(1)
      DO 11 I=2,68
      IF ( RSTAR(I) .LE. RLAMAX ) RLAMAX = RSTAR(I)
 11   CONTINUE
      RLAMAX = - RLAMAX
      PRINT, ' RLAMAX :',RLAMAX
C
C..... CONSTRUCTION OF THE PSTAR VECTOR .....
C
      DO 17 I=1,68
 17   PSTAR(I) = RSTAR(I)/RLAMAX
      DO 21 I=1,68
      IF ( PSTAR(I) .GT. 0.0 ) GO TO 21
      PSTAR(I) = 1.0 + PSTAR(I)
 21   CONTINUE
      PSTAR(1) = 0.0
      PSTAR(41) = 0.0
      PSTAR(55) = 0.0
      PSTAR(57) = 0.0
      PSTAR(58) = 0.0
      PSTAR(59) = 0.0
      PSTAR(60) = 0.0
      PSTAR(61) = 0.0
      PSTAR(62) = 0.0
      PSTAR(63) = 0.0
      PSTAR(64) = 0.0
      PSTAR(65) = 0.0
      PSTAR(66) = 0.0
      PSTAR(68) = 0.0
C
```

```
      ALG(1) = RLAMAX / ( RLAMAX - DLA(1) )
      ALG(2) = RLAMAX / ( RLAMAX - DLA(2) )
      ALG(3) = RLAMAX / ( RLAMAX - DLA(3) )
C
C..................... START CALCULATIONS K = 0 ..................
C
      K = 0
      TRSUM = 0.0
      DO 24 I=1,26
      PRO(I) = 0.0
   24 CONTINUE
      PRO(1) = DEXP( - ( DLA(1) * TO ))
      TRSUM = PRO(1)
      IF ( TRSUM .GE. ( 1.0 - ERROR )) GO TO 333
C ---------------------------------------------------------------------
      DO 26 I=1,4
      ST(I) = 0.0
      PRST(I) = 0.0
      LT(I) = 0.0
      PRLT(I) = 0.0
      FI(I) = 0.0
   26 COUN(I) = DLOG((RLAMAX - DLA(I)) * TO)
C ---------------------------------------------------------------------
      K = 1
   25 CONTINUE
      DO 27 I=1,10
   27 PIK(I) = 0.0
      CALL EVAL(NN,PHIOLD,PSTAR,TSTAR,PHINEW)
C
C..... CALCULATE THE ST(I,TO,K) QUANTITIES ....................
C
      IF ( K .GT. 1 ) GO TO 31
      DO 29 I=1,4
      LT(I) = - RLAMAX * TO
      IF ( LT(I) .GT. -120 ) THEN DO
      ST(I) = DEXP( LT(I) )
      ELSE DO
      ST(I) = 0.0
      END IF
   29 CONTINUE
      PRINT, ' ST(1,TO,1) :',ST(1)
      PRINT, ' ST(2,TO,1) :',ST(2)
      PRINT, ' ST(3,TO,1) :',ST(3)
      PRINT, ' ST(4,TO,1) :',ST(4)
      GO TO 33
   31 CONTINUE
      RK = K
      I = 1
      CALL SUM(I,RK,PRST,PRLT,COUN,TERM,LT,ST)
      I = 2
      RK = RK + 1.0
      CALL SUM(I,RK,PRST,PRLT,COUN,TERM,LT,ST)
```

```
      I = 3
      RK = RK + 1.0
      CALL SUM(I,RK,PRST,PRLT,COUN,TERM,LT,ST)
      I = 4
      RK = RK + 1.0
      CALL SUM(I,RK,PRST,PRLT,COUN,TERM,LT,ST)
   33 CONTINUE
C
      FI(1) = DXP1 - ST(1)
      FI(2) = DXP2 - ST(2)
      FI(3) = DXP3 - ST(3)
      FI(4) = 1.0 - ST(4)
C..... CALCULATE THE PIK ' S .....
      PIK(1) = 0.0
      CALL TWO(DLA,RLAMAX,ALG,K,FI,XXX)
      PIK(2) = XXX
      CALL THREE(ALG,K,FI,XXX)
      PIK(3) = XXX
      CALL FOUR(DLA,RLAMAX,PIK,XXX)
      PIK(4) = XXX
      CALL FIVE(ALG,K,FI,XXX)
      PIK(5) = XXX
      CALL SIX(FI,ALG,K,XXX)
      PIK(6) = XXX
      CALL SEVEN(FI,ALG,K,XXX)
      PIK(7) = XXX
      CALL EIGHT(FI,ALG,K,XXX)
      PIK(8) = XXX
      CALL NINE(ALG,K,FI,XXX)
      PIK(9) = XXX
      CALL TEN(FI,ALG,K,XXX)
      PIK(10) = XXX
C
C  ------------------------------------------------------------------
C  ..... EVALUATE THE CURRENT PROBABILITIES .....
C  ------------------------------------------------------------------
C
      DO 34 I=1,10
      IF ( PIK(I) .LE. XL ) THEN DO
      PIK(I) = 0.0
      ELSE DO
      PIK(I) = PIK(I)
      END IF
   34 CONTINUE
      DO 35 I=1,NN
      IF ( PHINEW(I) .LE. XL ) THEN DO
      PHINEW(I) = 0.0
      ELSE DO
      PHINEW(I) = PHINEW(I)
      END IF
   35 CONTINUE
```

```
      PRO(1) = PRO(1) + PHINEW(1) * PIK(1)
      DO 38 I=2,10
      PRO(I) = PRO(I) + PHINEW(I) * PIK(2)
   38 CONTINUE
      PRO(11) = PRO(11) + PHINEW(11) * PIK(3)
      DO 39 I=12,14
      PRO(I) = PRO(I) + PHINEW(I) * PIK(4)
   39 CONTINUE
      PRO(15) = PRO(15) + PHINEW(15) * PIK(5)
      DO 40 I=16,17
      PRO(I) = PRO(I) + PHINEW(I) * PIK(6)
   40 CONTINUE
      PRO(18) = PRO(18) + PHINEW(18) * PIK(7)
      DO 41 I=19,24
      PRO(I) = PRO(I) + PHINEW(I) * PIK(8)
   41 CONTINUE
      PRO(25) = PRO(25) + PHINEW(25) * PIK(9)
      PRO(26) = PRO(26) + PHINEW(26) * PIK(10)
C
C -----------------------------------------------------------------------
C ----------------  CHECK IF YOU ACIEVED THE TRUNCATION POINT ------
C -----------------------------------------------------------------------
      DO 44 I=1,10
   44 PSI(I) = 0.0
      PSI(1) = PHINEW(1)
      DO 45 I=2,10
   45 PSI(2) = PSI(2) + PHINEW(I)
      PSI(3) = PHINEW(11)
      DO 46 I=12,14
   46 PSI(4) = PSI(4) + PHINEW(I)
      PSI(5) = PHINEW(15)
      DO 47 I=16,17
   47 PSI(6) = PSI(6) + PHINEW(I)
      PSI(7) = PHINEW(18)
      DO 48 I=19,24
   48 PSI(8) = PSI(8) + PHINEW(I)
      PSI(9) = PHINEW(25)
      PSI(10) = PHINEW(26)
C
C
      DO 50 I=1,10
   50 TRSUM = TRSUM + PSI(I) * PIK(I)
      IF ( TRSUM .GE. ( 1.0 - ERROR )) GO TO 333
      K = K + 1
      DO 51 I=1,4
      PRST(I) = ST(I)
      PRLT(I) = LT(I)
   51 CONTINUE
      DO 55 I=1,NN
   55 PHIOLD(I) = PHINEW(I)
      GO TO 25
  333 CONTINUE
```

```
      SUMMA = 0.0
      DO 73 I=1,NN
      SUMMA = SUMMA + PRO(I)
   73 RLOPRO(I) = DLOG10(PRO(I))
      PRINT, ' SUM OF PROBABILITIES ',SUMMA
      WRITE(6,130)
  130 FORMAT(' '//' ',5X,' TSTAR ',20X,' RSTAR ',30X,' PSTAR ',//)
      DO 135 I=1,68
      WRITE(6,140) TSTAR(I),RSTAR(I),PSTAR(I)
  140 FORMAT(' ',7X,I4,7X,F28.20,6X,F28.20)
  135 CONTINUE
      WRITE(6,150) K
  150 FORMAT(' '//' ',' NUMBER OF TERMS SUMMED :',I7)
      WRITE(6,160) TO
  160 FORMAT(' '//' ',' TIME OF INTEREST :',F10.6)
      WRITE(6,162)
  162 FORMAT(' '/////' ',4X,' THE LOGARITHM OF THE PROBABILITIES IS : ')
      DO 165 I=1,NN
      WRITE(6,170) I,RLOPRO(I),PRO(I)
  170 FORMAT(' '//' ',5X,' LOGPRO(',I3,' ) = ',F30.26,5X,F30.26)
  165 CONTINUE
      STOP
      END
C
C-----------------------------------------------------------------------
C
      SUBROUTINE EVAL(NN,PHIOLD,PSTAR,TSTAR,PHINEW)
      DOUBLE PRECISION PHIOLD(26),PSTAR(68),PHINEW(26)
      INTEGER TSTAR(68)
      DOUBLE PRECISION PHIJ
      DATA XK/1.00E-62/
      DO 77 J=1,NN
      PHINEW(J) = 0.0
   77 CONTINUE
      I = 0
      DO 81 J=1,NN
      I = I + 1
      PHIJ = PHIOLD(J)
      PHINEW(J) = PHIJ * PSTAR(I) + PHINEW(J)
      MJ = TSTAR(I)
      IF ( MJ .EQ. 0) GO TO 81
      DO 79 KK=1,MJ
      I = I + 1
      LJK = TSTAR(I)
      IF ( PHIJ .LE. XK .OR. PSTAR(I) .LE. XK ) THEN DO
      PHINEW(LJK) = PHINEW(LJK)
      ELSE DO
      PHINEW(LJK) = PHINEW(LJK) + PHIJ * PSTAR(I)
      END IF
   79 CONTINUE
   81 CONTINUE
      RETURN
      END
```

```
C
C -----------------------------------------------------------------------
C
      SUBROUTINE TWO(DLA,RLAMAX,ALG,K,FI,XXX)
      DOUBLE PRECISION DLA(4),ALG(3),FI(4)
      DOUBLE PRECISION RLAMAX,XXX
      XXX = ALG(1)**K
      XXX = XXX * DLA(1) * FI(1) / RLAMAX
      RETURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE THREE(ALG,K,FI,XXX)
      DOUBLE PRECISION ALG(3),FI(4)
      DOUBLE PRECISION XXX
      XXX = ( ALG(2)**( K - 1)) * 3 * FI(2)
      XXX = XXX - ( ( ALG(1)**( K - 1 )) * 3 * FI(1) )
      RETURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE FOUR(DLA,RLAMAX,PIK,XXX)
      DOUBLE PRECISION DLA(4),PIK(10)
      DOUBLE PRECISION RLAMAX,XXX
      XXX = DLA(2) * PIK(3) / RLAMAX
      RETURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE FIVE(ALG,K,FI,XXX)
      DOUBLE PRECISION ALG(3),FI(4)
      DOUBLE PRECISION XXX
      IF ( K .GT. 1 ) GO TO 1005
      XXX = 0.0
      GO TO 1006
 1005 XXX = ( ALG(3)**(K-2)) * 3 * FI(3)
      XXX = XXX - ( ( ALG(2)**(K-2)) * 6 * FI(2) )
      XXX = XXX + ( ( ALG(1)**(K-2)) * 3 * FI(1) )
 1006 CONTINUE
      RETURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE SIX(FI,ALG,K,XXX)
      DOUBLE PRECISION FI(4),ALG(3)
      DOUBLE PRECISION XXX
      IF ( K .GT. 1 ) GO TO 1011
```

```
      XXX = 0.0
      RETURN
 1011 XXX = FI(4) - ( ( ALG(2)**(K-2) ) * 3 * FI(2) )
      XXX = XXX + ( ( ALG(1)**(K-2) ) * 2 * FI(1) )
      REYURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE SEVEN(FI,ALG,K,XXX)
      DOUBLE PRECISION FI(4),ALG(3)
      DOUBLE PRECISION XXX
      IF ( K .GT. 2 ) GO TO 1021
      XXX = 0.0
      RETURN
 1021 XXX = FI(4) - ( ( ALG(3)**(K-3) ) * 3 * FI(3) )
      XXX = XXX + ( ( ALG(2)**(K-3) ) * 3 * FI(2) )
      XXX = XXX - ( ( ALG(1)**(K-3) ) * FI(1) )
      RETURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE EIGHT(FI,ALG,K,XXX)
      DOUBLE PRECISION FI(4),ALG(3)
      DOUBLE PRECISION XXX
      XXX = FI(4) - ( ( ALG(1)**(K-1) ) * FI(1) )
      RETURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE NINE(ALG,K,FI,XXX)
      DOUBLE PRECISION ALG(3),FI(4)
      DOUBLE PRECISION XXX
      XXX = ( ( ALG(3)**(K-1) ) * 3 * FI(3) ) / 2
      XXX = XXX - ( ( ALG(1)**(K-1) ) * 3 * FI(1) ) / 2
      RETURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE TEN(FI,ALG,K,XXX)
      DOUBLE PRECISION FI(4),ALG(3)
      DOUBLE PRECISION XXX
      IF ( K .GT. 1 ) GO TO 1031
      XXX = 0.0
      RETURN
 1031 XXX = FI(4) - ( ( ALG(3)**(K-2) ) * 3 * FI(3) ) / 2
      XXX = XXX + ( ( ALG(1)**(K-2) ) * FI(1) ) / 2
      RETURN
      END
```

```
C
C -----------------------------------------------------------------------
C
      SUBROUTINE SUM(I,RK,PRST,PRLT,COUN,TERM,LT,ST)
      DOUBLE PRECISION PRST(4),PRLT(4),COUN(4),LT(4),ST(4)
      DOUBLE PRECISION RK,TERM
      RK = RK - 1.0
      CALL TRM(I,RK,PRLT,COUN,LT,TERM)
      ST(I) = PRST(I) + TERM
      RETURN
      END
C
C -----------------------------------------------------------------------
C
      SUBROUTINE TRM(I,RK,PRLT,COUN,LT,TERM)
      DOUBLE PRECISION PRLT(4),COUN(4),LT(4)
      DOUBLE PRECISION TERM,RK
      LT(I) = PRLT(I) + COUN(I) - DLOG(RK)
      IF ( LT(I) .GT. - 50.0 ) GO TO 500
      TERM = 0.0
      RETURN
  500 TERM = DEXP( LT(I) )
      RETURN
      END
```