SURE INPUT LANGUAGE

http://atb-www.larc.nasa.gov/ rwb/rel.html

October 27, 1999

Abstract

SURE is a reliability analysis program used for calculating upper and lower bounds on for the operational and death state probabilities for a large class of semi-Markov models. The program is especially suited for the analysis of fault-tolerant reconfigurable systems. The calculated bounds are close enough (usually within 5 percent of each other) for use in reliability studies of ultra-reliable computer systems. The SURE bounding theorems have algebraic solutions and are consequently computationally efficient even for large and complex systems. SURE can optionally regard a specified parameter as a variable over a range of values, enabling an automatic sensitivity analysis.

Contents

1 Introduction

The SURE program is a flexible, user-friendly reliability analysis tool. The program provides a rapid computational capability for semi-Markov models useful in describing the fault-handling behavior of fault-tolerant computer systems. The only modeling restriction imposed by the program is that the nonexponential recovery transitions must be fast in comparison to the mission time—a desirable attribute of all fault-tolerant systems. The SURE reliability analysis method utilizes a fast bounding theorem based on means and variances. These bounding theorems enable the calculation of upper and lower bounds on system reliability. The upper and lower bounds are typically within about 5 percent of each other. Since the computation method is extremely fast, large state spaces are not a problem.

This paper describes the input language for the SURE program. The reader is referred to [2] and [1] for a detailed description of the solution methods used by the SURE program.

1.1 Basic Program Concept

The user of the SURE program must describe his semi-Markov model to the SURE program using a simple language for enumerating all the transitions of the model. The SURE user must first assign numbers to every state in the system. The semi-Markov model is then described by enumerating all the transitions. There are two different statements used to enter transitions—one for slow transitions and the other for fast. If a transition is slow, then the following type of statement is used:

```
            1,2 = 0.0001;
```

This defines a slow exponential transition from state 1 to state 2 with rate 0.0001. The program does not require any particular units, e.g., hour^{-1} or sec^{-1}. However, the user must use consistent units(i.e. if the mission time is specified in hours, then the rates should be hour^{-1}). If the transition is fast, then either of two methods can be used to describe the transition: White's method and Lee's method.. The following specifies a fast transition using White's method

```
        2,4 = < 1E-4, 1E-6, 1.0 >;
```

The numbers in the brackets correspond to the conditional mean, conditional standard deviation, and transition probability of the fast transition, respectively. Using Lee's method the same transition would be specified as:

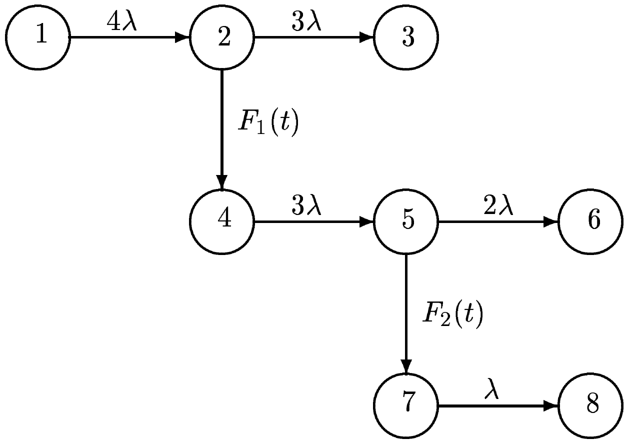
```
        @2 = < 1E-4, 1E-3, 0.99 >;
        2,4 = < 1.0 >;
```

The numbers in the brackets on the first line describe the holding time in state 2. The first number is the conditional mean. The next two numbers define a quantile of the fast distribution, i.e. the probability that the transition time is less than 1E-3 is 0.99. The number in the brackets on the second line is the probability that the transition from state 2 to state 4 succeeds over other competing fast transitions. Since there are no other competing transitions, this probability is 1.

Although the transition-description statements described above are the key constructs of the SURE language, the flexibility of the SURE program has been increased by adding several features commonly seen in programming languages such as FORTRAN or Pascal. In the next section, the SURE input language will be described in detail.

1.2 SURE Syntax By Way Of Example

The following semi-Markov model describes a TMR system with a spare:



This model assumes that the spare does not fail while inactive. The horizontal transitions represent fault arrivals. The coefficients of λ represent the number of processors in the configuration. The vertical transitions represent recovery from a fault. Since a TMR system uses 3-way voting for fault masking, there is a race between the occurence of fault #2 and removal of fault #1. If fault #2 wins the race, then the system fails (state 3). This model is described by the following SURE input file

```
LAMBDA = 1E-4;
MU1 = 2.7E-4;
SIGMA1 = 1.3E-3;
MU2 = 2.7E-4;
SIGMA2 = 1.3E-3;

1,2 = 4*LAMBDA;
2,3 = 3*LAMBDA;
2,4 = <MU1,SIGMA1>;
4,5 = 3*LAMBDA;
5,6 = 2*LAMBDA;
5,7 = <MU2,SIGMA2>;
7,8 = LAMBDA;
```

The first 5 statements equate values to identifiers (symbolic names). The identifier `LAMBDA` represents the processor failure rate. The identifiers `MU1` and `SIGMA1` are the mean and standard deviation of the time to replace a faulty processor with a spare. The identifiers `MU2` and `SIGMA2` are the mean and standard deviation of the time to degrade to a simplex. Conveniently, the only information SURE needs about the non-exponential recovery processes are the means and standard deviations. The final 7 statements define the transitions of the model. If the transition is a fault-arrival (or slow) the transition is assumed to be exponentially distributed and only the exponential rate need be provided. For example, the last statement defines a transition from state `7` to state `8` with a rate `LAMBDA`. If the transition is a recovery transition (or fast), the mean and standard deviation of the recovery time must be given. For example, the statement `2,4 = <MU1,SIGMA1>` defines a transition from state 2 to state 4 with mean recovery time `MU1` and standard deviation `SIGMA1`.

The following interactive session illustrates the solution of this model using SURE.

```
$ sure

  SURE V7.9   NASA Langley Research Center

  1? read trip1

  2:   LAMBDA = 1E-4;
  3:   MU1 = 2.7E-4;
  4:   SIGMA1 = 1.3E-3;
  5:   MU2 = 2.7E-4;
  6:   SIGMA2 = 1.3E-3;
  7:
  8:   1,2 = 4*LAMBDA;
  9:   2,3 = 3*LAMBDA;
 10:   2,4 = <MU1,SIGMA1>;
 11:   4,5 = 3*LAMBDA;
 12:   5,6 = 2*LAMBDA;
 13:   5,7 = <MU2,SIGMA2>;
 14:   7,8 = LAMBDA;

       0.02 SECS. TO READ MODEL FILE
 15? run

 MODEL FILE = trip1.mod                    SURE V7.9 22 Sep 97  11:24:02

                LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
  -----------   -----------   -----------   ---------------------------------
                2.25795e-09   2.32432e-09

 3 PATH(S) TO DEATH STATES
 0.000 SECS. CPU TIME UTILIZED
```

2 The SURE Input Language

The SURE input language includes two types of statements—model-definition statements and commands. These will be described in detail in the next sections.

2.1 Model-Definition Syntax

Models are defined in SURE by enumerating all of the transitions of the model.

2.1.1 Lexical Details

The state numbers must be positive integers between 0 and the `MAXSTATE` implementation limit, usually 1,000,000. (This limit can be changed by redefining a constant in the SURE program and recompiling the SURE source.) The transition rates, conditional means and standard deviations, etc., are floating point numbers. The Pascal REAL syntax is used for these numbers. Thus, all the following would be accepted by the SURE program:

```
0.001
12.34
1.2E-4
1E-5
```

The semicolon is used for statement termination. Therefore, more than one statement may be entered on a line. Comments may be included any place that blanks are allowed. The notation "(*" indicates the beginning of a comment and "*)" indicates the termination of a comment. The following is an example of the use of a comment:

```
LAMBDA = 5.7E-4;   (* FAILURE RATE OF A PROCESSOR *)
```

If statements are entered from a terminal (instead of by the `READ` command described below), then the carriage return is interpreted as a semicolon. Thus, interactive statements do not have to be terminated by an explicit semicolon unless more than one statement is entered on the line.

The SURE program prompts the user for input by a line number followed by a question mark. For example,

```
1?
```

The number is a count of the syntactically correct lines entered into the system thus far plus the current one.

2.1.2 Constant definitions

The user may equate numbers to identifiers. Thereafter, these constant identifiers may be used instead of the numbers. For example,

```
LAMBDA = 0.0052;

RECOVER = 0.005;
```

Constants may also be defined in terms of previously defined constants:

```
GAMMA = 10*LAMBDA;
```

In general, the syntax is

```
"name" = "expression";
```

where `"name"` discussed previously is a string of up to twelve letters, digits, and underscores (_) beginning with a letter, and `"expression"` is an arbitrary mathematical expression as described in a subsequent section entitled "Expressions".

2.1.3 Variable definition

In order to facilitate parametric analyses, a single variable may be defined. A range is given for this variable. The SURE system will compute the system reliability as a function of this variable. If the system is run in graphics mode (to be described later), then a plot of this function will be made. The following statement defines `LAMBDA` as a variable with range 0.001 to 0.009:

```
LAMBDA = 0.001 TO 0.009;
```

Only one such variable may be defined. A special constant, `POINTS`, defines the number of points over this range to be computed. The method used to vary the variable over this range can be either geometric or arithmetic and is best explained by example. Thus, suppose `POINTS` = 4, then

Geometric:

```
XV = 1 TO* 1000;
```

where the values of `XV` used would be 1, 10, 100, and 1000.

Arithmetic:

```
XV = 1 TO+ 1000;
```

where the values of `XV` used would be 1, 333, 667, and 1000.

The * following the `TO` implies a geometric range. A `TO+` or simply `TO` implies an arithmetic range.

One additional option is available—the `BY` option. By following the above syntax with `BY "increment"`, the value of `POINTS` is automatically set such that the value is varied by adding or multiplying the specified amount. For example,

```
V = 1E-6 TO* 1E-2 BY 10;
```

sets `POINTS` equal to 5 and the values of `V` used would be 1E-6, 1E-5, 1E-4, 1E-3, and 1E-2. The statement

```
Q = 3 TO+ 5 BY 1;
```

sets `POINTS` equal to 3, and the values of `Q` used would be 3, 4, and 5.

In general, the syntax is

```
"var" = "expression" TO {"c"} "expression" { BY "increment" }
```

where `"var"` is a string of up to twelve letters and digits beginning with a letter, `"expression"` is an arbitrary mathematical expression as described in the next section and the optional `"c"` is a + or *. The `BY` clause is optional; if it is used, then `"increment"` is any arbitrary expression.

2.1.4 Expressions

When specifying transition or holding time parameters in a statement, arbitrary functions of the constants and the variable may be used. The following operators[1] may be used:

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`**`	exponentiation

The following standard functions may be used:

`EXP(X)`	exponential function
`LN(X)`	natural logarithm
`SIN(X)`	sine function
`COS(X)`	cosine function
`SQRT(X)`	square root
`ABS(X)`	absolute value
`GAM(X)`	gamma function
`FACT(N)`	factorial of N.
`COMB(N,K)`	combinations of N things taken K at a time.
`PERM(N,K)`	permutations of N things taken K at a time.
`DIV(X,Y)`	integer quotient (operands are rounded first)
`MOD(X,Y)`	remainder (operands are rounded first)
`CYC(X,Y)`	cyclic-modulo (operands are rounded first)

Both `( )` and `[ ]` may be used for grouping in the expressions. The following are permissible expressions:

```
2E-4
1.2*EXP(-3*ALPHA);
7*ALPHA + 12*LAMBDA;
ALPHA*(1+LAMBDA) + ALPHA**2;
2*LAMBDA + (1/ALPHA)*[LAMBDA + (1/ALPHA)];
```

2.1.5 Slow transition description

A slow transition is completely specified by citing the source state, the destination state, and the transition rate. The syntax is as follows:

```
"source", "dest" = "rate";
```

[1]Please note that the associativity of the exponentiation operator was inconsistant between versions of SURE/STEM/PAWS prior to version 7.9.8 and FTC version 2.8.2 and the new version of ASSIST. Users of ASSIST 7.0 and higher should use SURE/STEM/PAWS 7.9.8 or higher and FTC 2.8.2 or higher.

The prior releases of SURE/STEM/PAWS/FTC used left-to-right associativity whereas the new versions of these programs and ASSIST 7.0 use the more common right-to-left associativity. For example, consider: `VAL = 1.1 ** 1.883 ** 1.448`. In the prior releases of SURE/STEM/PAWS/FTC, this was equivalent to `VAL = (1.1 ** 1.883) ** 1.448`. Whereas in ASSIST 7.0, and the version 7.9.8 of SURE/STEM/PAWS/FTC it is equivalent to: `VAL = 1.1 ** (1.883 ** 1.448)`.

where "source" is the source state, "dest" is the destination state, and "rate" is any valid expression defining the exponential rate of the transition. The following are valid SURE statements:

```
PERM = 1E-4;
TRANSIENT = 10*PERM;
1,2 = 5*PERM;
1,9 = 5*(TRANSIENT + PERM);
2,3 = 1E-6;
```

2.1.6 Fast transition description

To enter a fast transition, the SURE user may use either of two methods: White's method or Lee's method. White's method is preferred by most users.

White's method: The following syntax is used for White's method.:

```
"source" , "dest"  = < "mu", "sig" {, "frac" } >;
```

where

"mu"	=	an expression defining the conditional mean transition time, m(F*)
"sig"	=	an expression defining the conditional standard deviation of the transition time, s(F*)
"frac"	=	an expression defining the transition probability, r(F*)

and "source" and "dest" define the source and destination states, respectively. The third parameter "frac" is optional. If omitted, the transition probability is assumed to be 1.0, i.e., only one fast transition.

All the following are valid (while in White's mode):

```
2,5 = <1E-5, 1E-6, 0.9>;

THETA = 1E-4;
5,7 = <THETA, THETA*THETA, 0.5>;
7,9 = <0.0001,THETA/25>;
```

Lee's method: To describe a fast transition using Lee's method, the following syntax is used.:

```
"source" , "dest" = ;

@ "source"  =  < "hmu", "quant" {, "prob" } >;
```

where

"source"	=	source state
"dest"	=	destination state
"frac"	=	an expression defining the transition probability
"hmu"	=	an expression defining the conditional mean fast transitions' holding time in the state, given that holding time is less than "quant"
"quant"	=	an expression defining the percentile or censoring point
"prob"	=	an expression defining the probability that the holding time in the state is less than "quant"

All the following are valid SURE statements (while in the Lee mode):

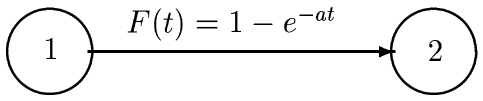
```
5,6 = <0.5>;
FRACT = 0.0 TO 0.5;
5,7 = <FRACT>;
5,8 = <0.5 - FRACT>;
@5 = <0.00034, 0.003, (1.0-1E-4) >;
```

Although there may be many fast transitions from a state, the "`@source`" statement should be issued only once for the state.

The SURE user must decide which method he will use before entering his model. Either the Lee method or White method may be used to describe the model, but both cannot be used at the same time. By default, the program assumes that the White method will be used. If Lee's method is desired, the `LEE` command must be issued prior to entering any fast transition.

2.1.7 FAST exponential transition description

Often when performing design studies, experimental data is unavailable for the fast processes of a system. In this case, one must assume some properties of the underlying processes. For simplicity, these fast transitions are often assumed to be exponentially distributed. However, it is still necessary to supply the conditional mean and standard deviation to the SURE program since they are fast transitions. If there is only one fast transition from a state, then these parameters are easy to determine. Suppose we have a fast exponential recovery from state 1 to state 2 with unconditional rate a:



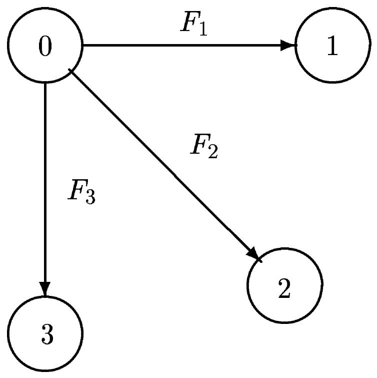
The SURE input is simply

```
1,2 = <1/a, 1/a, 1>;
```

In this case, the conditional mean and standard deviation are equivalent to the unconditional mean and standard deviation. The above transition can be specified using the following syntax:

```
1,2 = FAST a;
```

When multiple recoveries are present from a single state, then care must be exercised to properly specify the conditional means and standard deviations required by the SURE program. Suppose we have the model:



where

$$F_1(t) = 1 - e^{-at}$$
$$F_2(t) = 1 - e^{-bt}$$
$$F_3(t) = 1 - e^{-ct}$$

The SURE input describing the above model section is:

```
0,1 = < 1/(a+b+c), 1/(a+b+c), a/(a+b+c) >;
0,2 = < 1/(a+b+c), 1/(a+b+c), b/(a+b+c) >;
0,3 = < 1/(a+b+c), 1/(a+b+c), g/(a+b+c) >;
```

Note that the conditional means and standard deviations are not equal to the unconditional means and standard deviations (e.g. `1/a`.)

The following can also be used to define the above model:

```
0,1 = FAST a;
0,2 = FAST b;
0,3 = FAST c;
```

The SURE program automatically calculates the conditional parameters from the unconditional rates a,b and c. The FAST exponential capability can only be used in conjunction with the WHITE method of specifying recovery transitions. The user may mix FAST exponential transitions with other general transitions. However, care must be exercised in specifying the conditional parameters of the non-exponential fast recoveries in order to avoid inconsistencies.

2.2 SURE Commands

Two types of commands have been included in the user interface. The first type of command is initiated by one of the following reserved word:

```
EXIT    READ    READO   INPUT    LEE     RUN

SHOW    IF      CALC    ORPROB   PLOT
```

The second type of command is invoked by setting one of the following special constants

```
AUTOFAST  ECHO    LIST    POINTS  PRUNE   QTCALC  START

TIME      TRUNC   WARNDIG
```

equal to one of its pre-defined values.

2.2.1 EXIT command

The EXIT command causes termination of the SURE program.

2.2.2 READ/READ0 command

A sequence of SURE statements may be read from a disk file. The following interactive command reads SURE statements from a disk file named `SIFT.MOD`:

```
READ SIFT.MOD;
```

If no file name extent is given, the default extent `.MOD` is assumed. A user can build a model description file using a text editor and use this command to read it into the SURE program.

The `READ0` has been added for convenience. The function of `READ0` is the same as `READ` except it sets `ECHO = 0` before processing the file. Thus, `READ0 "file";` is equivalent to:

```
ECHO = 0; READ "file";
```

2.2.3 INPUT command

This command increases the flexibility of the `READ` command. Within the model description file created with a text editor, `INPUT` commands can be inserted that will prompt for values of specified constants while the model file is being processed by the `READ` command. For example, the command

```
INPUT LVAL;
```

will prompt the user for a number as follows:

```
LVAL?
```

and a new constant `LVAL` is created that is equal to the value input by the user. Several constants can be interactively defined using one statement, for example:

```
INPUT X, Y, Z;
```

2.2.4 RUN command

After a semi-Markov model has been fully described to the SURE program, the `RUN` command is used to initiate the computation:

```
RUN;
```

The output is displayed on the terminal according to the `LIST` option specified. If the user wants the output written to a disk file instead, the following syntax is used:

```
RUN "outname";
```

where the output file `"outname"` may be any permissible VAX VMS file name. Two positional parameters are available on the `RUN` command. These parameters enable the user to change the value of the special constants `POINTS and LIST` in the `RUN` command. For example

```
RUN (30,2) OUTFILE.DAT
```

is equivalent to the following sequence of commands:

```
POINTS = 30;

LIST = 2;

RUN OUTFILE.DAT
```

Each parameter is optional so the following are acceptable:

```
    RUN(10);         -- change POINTS to 10 then run.

    RUN(,3);         -- change LIST to 3 and run.

    RUN(20,2);       -- change POINTS to 20 and LIST to 2 then run.
```

After a run is completed, the SURE program clears all of the transition, constant and variable definitions, returning the program state to its original state. However, throughout the session, the output of each `RUN` is stored internally. The results of prior runs are available in special variables which can be referenced in future model descriptions or in a `CALC` command. The syntax is as follows:

`#L1`	–	lowerbound for run #1 (no variable)
`#U2`	–	upperbound for run #2 (no variable)
`#1`	–	upperbound for run #1 (no variable)
`#L1[3]`	–	lowerbound for third value of variable on run #1
`#U2[1]`	–	upperbound for first value of variable on run #2

2.2.5 ECHO constant

The `ECHO` constant can be used to turn off the echo when reading a disk file. The default value of `ECHO` is 1, which causes the model description to be listed as it is read. (See example 3 in the section entitled "Example SURE Sessions.")

2.2.6 LIST constant

The amount of information output by the program is controlled by this command. Four list modes are available as follows:

`LIST = 0;`	No output is sent to the terminal, but the results can still be displayed using the PLOT command.
`LIST = 1`	Only the upper and lower bounds on the probability of total system failure are listed. This is the default.
`LIST = 2`	The probability bounds for each death state in the model are reported along with the totals.
`LIST = 3`	The probability for the operational states is reported in addition to the death states.
`LIST = 4`	Every path in the model is listed and its probability of traversal. The probability bounds for each death state in the model is reported along with the totals.

If a variable is defined and `LIST=1` is specified, then the summary statistics are only given for the value of the variable for which the bounds had the worst accuracy. If `LIST >= 2` then the summary statistics are given for each value of the variable.

2.2.7 POINTS constant

The `POINTS` constant specifies the number of points to be calculated over the range of the variable. The default value is 25. If no variable is defined, then this specification is ignored.

2.2.8 START constant

The `START` constant is used to specify the start state of the model. If the `START` constant is not used, the program will use the source state (i.e. the state with no transitions into it) of the model (if one exists.) If there is no source state in the model, the program will use the first state entered as the start state. If no start state is specified and there are two or more source states, an error message is issued. The program arbitrarily chooses one of the source states as the start state and proceeds.

2.2.9 TIME constant

The `TIME` constant specifies the mission time. For example, if the user sets `TIME = 1.3`, the program computes the probability of entering the death states of the model within time `1.3`. The default value of `TIME` is `10`. All parameter values must be in the the same units as the `TIME` constant.

2.2.10 ORPROB command

A common complaint about the Markov approach to modeling is the rapid growth in state space size as the complexity of a system is increased. For large, complex inter-dependent systems, this is often unavoidable. But, systems which consist of several isolated subsystems can be analyzed easily using the additive law of probability.

Suppose the probabilities that subsystem 1 and subsystem 2 fail within the mission time are `P1` and `P2`, respectively. If these subsystems fail independently, the probability of system failure, `Psys`, can be calculated as follows:

```
Psys = P1 + P2 - (P1)(P2).
```

If there are failure dependencies between the subsystems, then a single model must be used.

The `ORPROB` command lists all of the previous run output results and then computes the probabilistic OR of the previous runs. See example 8 in the section entitled "Examples". The `PLOT` command may be used to plot the results of the `ORPROB` command. If the variable feature of SURE is used and `LIST = 1`, then the `ORPROB` command does not list out the answers from the previous runs. Only the probabilistic OR for each value of the variable is given. If `LIST = 2` is set prior to issuing `ORPROB`, then a detailed list of all the outputs from the previous runs, along with the probabilistic OR of the runs for each value of the variable, is given.

2.2.11 PLOT command

In SURE Version V7.9.9 a primitive but usable interface to the GNUPLOT plotting package was added. The GNUPLOT package is publically available at

```
http://www.cs.dartmouth.edu/gnuplot_info.html
```

After issuing a `RUN` command, the SURE user can issue a `PLOT` command:

```
? PLOT
```

and the SURE program will write out files that can used by GNUPLOT to plot the upper and lower bounds obtained from the `RUN` command. GNUPLOT is run from a Unix prompt as follows:

```
$ gnuplot

     G N U P L O T
     Linux version 3.5 (pre 3.6)

gnuplot> load 'sure.gp'
```

The file "`sure.gp`" is one of three files written by the sure program. The SURE program closes the `sure.gp` file as soon as the PLOT command completes, so that GNUPLOT can run immediately after the PLOT command in a separate X window.

The `PLOT` command may be followed by any of the following options:

`XLOG`	makes x scale logarithmic
`YLOG`	makes y scale logarithmic
`XYLOG`	makes xy scale logarithmic

For example:

```
? PLOT XYLOG
```

2.2.12 QTCALC constant

The value of the `QTCALC` constant determines the numerical method used to compute Q(T) (See SURE TP). If `QTCALC = 0`, the program uses White's algebraic formulas for Q(T). If `QTCALC = 1`, the program uses an exponential matrix solver to calculate Q(T) rather than the algebraic approximations. This method is slower but is much more accurate when the mission time is long. The default value of `QTCALC` is 2, which specifies that the program should automatically select the appropriate Q(T) algorithm on a path-by-path basis. The SURE program indicates when the exponential matrix solver is used by writing `<ExpMat>` in the comments field of the output.

2.2.13 Pruning and Loop Truncation Method

The SURE program follows paths until they reach a death state or the probability drops below the current "prune" level. This level is determined automatically by default (See `AUTOPRUNE` constant), but can also be set manually using the `PRUNE` constant. The error due to pruning is always added to the upper bound.

This techniques is also used to truncate loops. SURE continues to "unfold" the loop until the resulting paths are producing insignificant amounts of probability based on the current `PRUNE` level. An error bound on this truncation is computed. The error due to pruning and loop truncation are added together and reported in the comments field as follows when `LIST=0`.

```
                  <prune 1.2e-12>
```

If `LIST` is set to 2 or more, the prune and trunc error bounds are listed on a separate row as follows:

```
 DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                RUN #4
 ----------   -----------   -----------    ---------------------------------
      1       9.19500E-12   1.00000E-11
      4       3.46542E-10   4.77867E-10
 sure prune   0.00000e+00   1.00000E-13
              -----------   -----------
   SUBTOTAL   3.475645-10   4.87797E-10
```

The row that begins `sure prune` reports an upper bound on the error due to pruning and truncation. The pruning error is *added to the upperbound.* The upper bound is consequently always an upper bound on the probability of system failure, even if pruning is too severe. If the prune level is too severe, then the bounds will be far apart, but valid. See example 11 for an illustration of pruning.

2.2.14 A Warning About Fast Loops

There is one situation where the SURE program's automatic loop truncation method gets into trouble: fast loops. Models that contain fast loops, i.e. loops with only fast transitions, can cause the program to run forever unless a "safety value" is used. Fast loops generate an infinite sequence of paths which do not decrease in probability (as far as SURE's Upper Bound is concerned). Thus, the program would run forever if only pruning were invoked. The `TRUNC` command will terminate the expansion of a loop after a specified number of times. The default value is `25` which will never be hit for most models. However, for models containing fast loops, it will keep the program from running forever. If a positive value of `TRUNC` is given, the program stops the current loop but continues processing. If a negative value is specified, the program terminates when a loop is encountered which does not get pruned before it is unfolded `ABS(TRUNC)` times.

2.2.15 AUTOPRUNE constant

The default value of `AUTOPRUNE` is `1`. This directs the SURE program to automatically determine a level of pruning. The program will select a prune level based on the probability of the first death state it encounters. As more death states are encountered, the program updates the value of `PRUNE`. The `PRUNE` level is updated after the following number of death states are reached: `1, 10, 100, 1000`, etc. The last (and thus highest) level of pruning is reported in the comments field. See Example 11 for an illustration.

To turn off autopruning, the user enters:

```
AUTOPRUNE = 0;
```

before the `RUN` command.

2.2.16 PRUNE constant

The SURE user can manually specify the level of pruning desired using the `PRUNE` constant. A path is traversed by the SURE program until the probability of reaching the current point on the path falls below the pruning level. For example, if `PRUNE = 1E-14` and the upper bound falls below `1E-14` at any point on the path, the analysis of the path is terminated and its contribution to the death state probabilities is added to the upper bound. For very large models, it is recommended that the user start with a very large value of `PRUNE` (e.g. `1E-10`) and decrease the value (e.g. to `1E-15`) until the `PRUNING TOO SEVERE` message disappears.

2.2.17 Initialization of State Probabilities

The SURE user can specify initial probability to more than 1 state. In early versions of SURE, all of the initial probability was assigned to 1 state using the `START` statement. Since version , the user can distribute the initial probability over as many states as he desires. The `INITIAL_PROBS` statement is used to do this. For example,

```
      INITIAL_PROBS(1: 0.3, 2: 0.7);
```

assigns an initial probability of 0.3 to state 1 and an initial probability of 0.7 to state 2. The sum of all of the probabilities must add to 1. The user may also specify upper and lower bounds on the initial state probabilities:

```
      INITIAL_PROBS(1: (0.27,0.31), 2: (0.69,0.71));
```

When `LIST >= 3` the program writes out a file containing all of the state's probabilities in exactly the format needed to initialize the states in a subsequent run. The name of the file is obtained from the last file read in, by replacing the `.mod` with `.ini`. For example, if the last file read in were `amodel.mod`, then the file would be named `amodel.ini` and, if the last file read in were `amodel.MOD`, then the file would be named `amodel.ini`. The `.ini` extent is never capitalized in SURE.

2.2.18 Computing Operational State Probabilities

SURE will report the operational state probabilities as well as the death state probabilities when the `LIST` variable is set to 3 or above.

The program also allows the user to initialize a model using these same operational state probabilities (See Section 2.2.17). These features support the use of sequences of reliability models to model systems with phased missions or non-constant failure rates. This is discussed in Appendix A.

2.2.19 SHOW command

The value of an constant or variable may be displayed by the following command:

```
       SHOW ALPHA;
```

Information about a transition may also be displayed by the `SHOW` command. For example, information concerning the transition from state `654` to state `193` is displayed by the following command:

```
       SHOW 654-193;
```

If the model is described using Lee's method, the information about a state holding time may be displayed. For example, state `12` holding time characteristics are listed in response to

```
       SHOW 12;
```

More than one constant, variable, etc. may be shown at one time:

```
       SHOW ALPHA, 12-13, BETA, 123;
```

2.2.20 IF command

The IF statement provides a "conditional assembly" capability to the SURE program. The statement following the `THEN reserved` word is only processed if the preceding boolean expression is true. The syntax of this statement is:

```
IF "expression" "bool-op" "expression" THEN "statement";
```

where `"bool-op"` is one of the following operators: `=`   `>=`.

The following session illustrates this command:

```
$ SURE

  1? X = 1; Y = 2;

  2? IF X = 1 THEN Y = 3;
     Y       CHANGED TO 3.00000E+00

  3? SHOW Y;
     Y        = 3.00000E+00

  4? IF Y > X THEN 1,2 = 1E-4;

 5? SHOW 1-2;

     TRANSITION 1 -> 2: EXPONENTIAL RATE = 1.00000E-4;

  6? IF X   7? SHOW 2-3

     TRANSITION 2 -> 3 NOT FOUND

  8? EXIT
```

2.2.21 CALC command

For convenience, a calculator function has been included. This command allows the user to obtain the value of an arbitrary expression. For example, if the following commands are entered:

```
X = 1.6E-1;

CALC  (X-.12)*EXP(-0.001) + X**3;
```

the system responds with:

```
      = 4.405601999335E-02
```

If a variable has been defined prior to issuing the `CALC` function, the expression is computed as a function of the variable over the specified range. The `PLOT` command can be used after the `CALC` command to obtain a plot of the function. This feature is illustrated in example 10 of the section entitled "Examples". The output can be sent to a disk file instead of the terminal y using the following syntax:

```
   CALC "expression" TO "filename";
```

where "filename" is the name of the destination file.

2.2.22 LINEAR command

The LINEAR command allows the user to specify that a model will be linear. Linear models can be solved more efficiently by SURE. If the LINEAR command is present and the model turns out to be non-linear, an appropriate message will be displayed before returning control to the operating system. A model is linear if there is no SURE variable or if all "expressions" are linear functions of the variable, e.g., 3*LAMBDA where LAMBDA = 1E-6 TO* 1E-4 BY 10.

2.2.23 AUTOFAST constant

If the special constant AUTOFAST is set equal to 1, then the reserved word FAST does not have to be used before a rate expression to indicate that the transition is fast. The program automatically decides if the rate is fast with respect to the mission time. If the product of the transition rate and the mission time TIME is greater than 100 then the transition is treated as FAST and the conditional means and standard deviations are automatically calculated just as if FAST had been explicitly specified. Otherwise, the transition is treated as a slow transition. The default value of AUTOFAST is 0 which implies no automatic conversion to FAST.

2.2.24 LEE command

The LEE command prepares the program to receive fast transition commands according to the Lee syntax. By default, the program expects fast transitions to be described in White format. The syntax of the LEE command is

```
        LEE;
```

The LEE command must be issued prior to entering any fast transitions. The FAST exponential syntax cannot be used in LEE mode.

2.3 ASSIST PRUNE States

The ASSIST program generates models for SURE [3]. The ASSIST program has the capability to prune models and records this in a manner that can be reported by SURE. The SURE program reports the ASSIST prune states separately from the death states as follows:

```
DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #4
----------   -----------   -----------    ---------------------------------
     8       9.99500E-12   1.00000E-11
     7       1.66542E-10   1.66667E-10
    10       9.99500E-14   1.00000E-13
             -----------   -----------
  SUBTOTAL   1.76637E-10   1.76767E-10

PRUNESTATE    LOWERBOUND    UPPERBOUND
----------   -----------   -----------
PRUNE   3    9.99500E-15   1.00000E-14
PRUNE  11     9.99500E-18   1.00000E-17
              -----------   -----------
  SUBTOTAL    1.00050E-14   1.00100E-14


TOTAL         1.76637E-10   1.76777E-10
```

Note that in the `TOTAL` line, the upper bound includes the contribution of the prune states whereas the lower bound does not. Thus, the `TOTAL` lines are valid bounds on the system failure probability. If the PRUNE-state upper bound is significant with respect to the `TOTAL` upper bound, then the user has probably pruned his model too severely. The upper and lower bounds can be made significantly closer by relaxing the amount of pruning. The ASSIST program wrote the following into the SURE input file to inform the SURE program which states are ASSIST-level prune states:

```
    PRUNESTATES = (3,11);
```

One final point, please note that there are two kind of pruning: SURE-level pruning and ASSIST-level pruning. ASSIST-level pruning is done at model generation time. After model building is completed, the amount of processing time can be reduced using SURE-level pruning. This is invoked by the SURE command `PRUNE = "rate"` or the default autopruning (See `AUTOPRUNE` command). SURE-level pruning often will still be effective in conjunction with ASSIST-level pruning.

3 The STEM and PAWS Programs

The STEM (Scaled Taylor Exponential Matrix) and PAWS (Pade Approximation With Scaling) programs are automatic Markov model solvers. They use the exact same input language as the SURE program and compute the death state probabilities under the assumption that the recovery distributions are exponential. Therefore, a SURE model description file can be read without modification by other programs. The slow exponential transitions are interpreted the same in PAWS and STEM as in SURE. However, the inputs defining the standard deviation of the fast recoveries are ignored. This is necessary since the standard deviation of an exponential distribution is equal to the mean. Thus, the following SURE input command

```
1,2 = <MU,STD>;
```

is interpreted as

```
1,2 = 1/MU;
```

If there is more than 1 transition from a state as below:

```
1,2 = <MU1,STD1,P1>;
1,3 = <MU2,STD2,P2>;          (* note: P1 + P2 = 1 *)
```

then these are interpreted as

```
1,2 = P1/MU1;
1,3 = P2/MU2;
```

4 New Model Display Capability: MODDISP

A new program, MODDISP, has been provided that translates a sure model into a format that can be graphically displayed using the publically available tool VCG. The VCG tool can be obtained from

```
http://www.cs.uni-sb.de/RW/users/sander/html/gsvcg1.html
```

or

```
ftp://ftp.cs.uni-sb.de/pub/graphics/vcg/vcg.tgz
```

To graphically display a SURE model, the following is issued at a Unix prompt:

```
$ moddisp < sure-model.mod | xvcg -
```

If the input model was generated by ASSIST, the nodes are labelled with the ASSIST state vectors. If the SURE state numbers are preferred, the following can be used:

```
$ moddisp -s < sure-model.mod | xvcg -
```

All of the capabilities of VCG can be exploited by generating a .vcg file:

```
$ moddisp < sure-model.mod > sure-model.vcg
```

and editing it.

5 New Plotting Capability

In SURE Version V7.9.9 a primitive but usable interface to the GNUPLOT plotting package was added. The GNUPLOT package is publically available at

```
http://www.cs.dartmouth.edu/gnuplot_info.html
```

After issuing a `RUN` command, the SURE user can issue a `PLOT` command:

```
? PLOT
```

and the SURE program will write out files that can used by GNUPLOT to plot the upper and lower bounds obtained from the `RUN` command. GNUPLOT is run from a Unix prompt as follows:

```
$ gnuplot

     G N U P L O T
     Linux version 3.5 (pre 3.6)

gnuplot> load 'sure.gp'
```

The file "`sure.gp`" is one of three files written by the SURE program when the PLOT command is issued. Two other files "`LowerBound.dat`" and "`'UpperBound.dat`" contain the SURE run data. Alternatively the user can issue the following commands to GNUPLOT:

```
set ylabel 'Prob Failure'
plot 'LowerBound.dat' with lines, 'UpperBound.dat' with lines
```

The SURE program closes the three plot output files as soon as the PLOT command completes, so that GNUPLOT can run immediately after the PLOT command in a separate X window.

The `PLOT` command may be followed by any of the following options:

`XLOG`	makes x scale logarithmic
`YLOG`	makes y scale logarithmic
`XYLOG`	makes xy scale logarithmic

For example:

```
? PLOT XYLOG
```

This causes `set logscale xy` to also be written to the "`sure.gp`" file.

6 Examples SURE Sessions

6.1 Outline of a Typical Session

The SURE program was designed for interactive use. The following method of use is recommended (See example 2.)

1. Create a file of SURE commands using a text editor describing the semi-Markov model to be analyzed.
2. Start the SURE program and use the `READ` command to retrieve the model information from this file.
3. Then, various commands may be used to change the values of the special constants, such as `LIST`, `POINTS`. `QTCALC`, `TRUNC`, etc., as desired. Altering the value of a constant identifier does not affect any transitions entered previously even though they were defined using a different value for the constant. The range of the variable may be changed after transitions are entered.
4. Enter the `RUN` command to initiate the computation.

The following examples illustrate interactive SURE sessions. For clarity, all user inputs are given in lower-case letters.

6.2 Example 1

This session illustrates direct interactive input and the type of error messages given by SURE:

```
$ sure

  SURE V7.5    NASA Langley Research Center

  1? lambda = 1e-5;
  2? 1,2 = 6*lambda;
  3? 2,3 = 5*lamba;
                  ^ IDENTIFIER NOT DEFINED
  3? 2,3 = 5*lambda;
  4? show 2-3;
     TRANSITION 2 -> 3: RATE = 5.00000E-5
  5? 2,4 = ;
  6? 4,5 = 2*lambda;
  7? list = 2;
  8? time = 1;
  9? run

 DEATHSTATE      LOWERBOUND    UPPERBOUND    COMMENTS    RUN #1
 ----------      ----------    ----------    --------------------------
        3       2.93992E-13   3.00000E-13
        5       5.95908E-10   6.00000E-10

 TOTAL          5.96202E-10   6.00300E-10

 *** WARNING: SYNTAX ERRORS PRESENT BEFORE RUN
 2 PATH(S) PROCESSED
 0.100 SECS. CPU TIME UTILIZED

 10? exit
```

The warning message indicates that a syntax error was encountered by the program.

If a user receives this message, he should check his input file to make sure that the model description is correct. In this example, since the syntax error was corrected in the next line, the model was correct. A complete list of program-generated error messages is given in the Appendix B.

Since `LIST = 2`, upper and lower bounds are given for each death state as well as the total. The mission time is set to 1 in statement 8. If this statement were omitted, the program would use 10 by default.

6.3 Example 2

The following session indicates the normal method of using SURE. Prior to this session, a text editor has been used to build file `TRIADP1.MOD`. This file contains a description of a triad system with one spare. The system uses 3-fold redundancy to mask single processor faults. If a spare is available the system replaces a faulty processor with the spare. If no spare is available the system degrades to a simplex. For simplicity the means and standard deviations of both types of recovery are assumed to be the same— `RECOVER` and `STDEV` respectively. The program displays the contents of the files as it is read (with the `READ` command). Input lines which are read, are labeled with a line number followed by a colon.

```
>
$sure

  SURE V7.9   NASA Langley Research Center
```

```
 1? read triadp1

 2: LAMBDA = 1E-6 TO* 1E-2;
 3: RECOVER = 2.7E-4;
 4: STDEV = 1.3E-3;
 5: 1,2 = 3*LAMBDA;
 6: 2,3 = 2*LAMBDA;
 7: 2,4 = <RECOVER,STDEV>;
 8: 4,5 = 3*LAMBDA;
 9: 5,6 = 2*LAMBDA;
10: 5,7 = <RECOVER,STDEV>;
11: 7,8 = LAMBDA;
12: POINTS = 10;
13: TIME = 6;

     0.02 SECS. TO READ MODEL FILE
14? run

MODEL FILE = triadp1.mod                  SURE V7.9 22 Sep 97  13:46:43


   LAMBDA      LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
 -----------   -----------   -----------   ---------------------------------
 1.00000e-06   9.40296e-15   1.00441e-14
 2.78256e-06   7.71327e-14   8.22406e-14
 7.74264e-06   6.90469e-13   7.33126e-13
 2.15443e-05   7.35487e-12   7.75250e-12
 5.99484e-05   1.00201e-10   1.04754e-10
 1.66810e-04   1.70631e-09   1.77475e-09
 4.64159e-04   3.31737e-08   3.45028e-08
 1.29155e-03   6.81859e-07   7.14439e-07
 3.59381e-03   1.41321e-05   1.51683e-05
 1.00000e-02   2.83744e-04   2.92932e-04   <ExpMat>

3 PATH(S) TO DEATH STATES
Q(T) ACCURACY >= 14 DIGITS
0.001 SECS. CPU TIME UTILIZED
```

The following interactive session illustrates the use of the ECHO constant. This constant is used when the model description file is large and one desires that the model input not be listed on the terminal as it is read by the SURE program.

```
$sure

  SURE V7.9    NASA Langley Research Center

  1? echo = 0;
  2? read ftmp2.mod;
```

```
 26? run
 MODEL FILE = ftmp2.mod                    SURE V7.9 22 Sep 97  13:49:44

    LAMBDA       LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
  -----------   -----------   -----------    ---------------------------------
  1.00000e-04   4.88265e-10   5.02254e-10    <prune 7.3e-17>
  2.00000e-04   1.95291e-09   2.01808e-09    <prune 4.7e-15>
  3.00000e-04   4.39357e-09   4.56117e-09    <prune 5.3e-14>
  4.00000e-04   7.80964e-09   8.14545e-09    <prune 3.0e-13>
  5.00000e-04   1.22003e-08   1.27852e-08    <prune 1.1e-12>
  6.00000e-04   1.75646e-08   1.84953e-08    <prune 3.4e-12>
  7.00000e-04   2.39013e-08   2.52827e-08
  8.00000e-04   3.12090e-08   3.31707e-08
  9.00000e-04   3.94859e-08   4.21702e-08
  1.00000e-03   4.87302e-08   5.22958e-08

 7 PATH(S) TO DEATH STATES 1 PATH(S) PRUNED
 HIGHEST PRUNE LEVEL =  3.60000e-12
 0.016 SECS. CPU TIME UTILIZED
 27? exit
```

6.4 Example 3

This example illustrates the use of SURE to solve a model generated by ASSIST[3].

```
> assist triadcold.ast
ASSIST VERSION 7.1                   NASA Langley Research Center
PARSING TIME = 0.07 sec.
generating SURE model file...
RULE GENERATION TIME = 0.00 sec.
NUMBER OF STATES IN MODEL = 13
NUMBER OF TRANSITIONS IN MODEL = 24
6 DEATH STATES AGGREGATED INTO STATE 1

> sure

  SURE V7.9   NASA Langley Research Center

  1? read triadcold

  2: N_PROCS = 3;
  3: N_SPARES = 2;
  4: LAMBDA_P = 1E-4;
  5: LAMBDA_S = 1E-5;
  6: DELTA = 3.6E3;
  7:
```

```
  8:
  9:      2(* 3,0,2,0 *),     3(* 3,1,2,0 *)        = (3-0)*LAMBDA_P;
 10:      2(* 3,0,2,0 *),     4(* 3,0,2,1 *)        = 2*LAMBDA_S;
 11:      3(* 3,1,2,0 *),     5(* 3,0,1,0 *)        = FAST (1-(0/2))*1*DELTA;
 12:      3(* 3,1,2,0 *),     1(* 3,2,2,0 DEATH  *) = (3-1)*LAMBDA_P;
 13:      3(* 3,1,2,0 *),     6(* 3,1,2,1 *)        = 2*LAMBDA_S;
 14:      4(* 3,0,2,1 *),     6(* 3,1,2,1 *)        = (3-0)*LAMBDA_P;
 15:      4(* 3,0,2,1 *),     7(* 3,0,2,2 *)        = 2*LAMBDA_S;
 16:      5(* 3,0,1,0 *),     8(* 3,1,1,0 *)        = (3-0)*LAMBDA_P;
 17:      5(* 3,0,1,0 *),     9(* 3,0,1,1 *)        = 1*LAMBDA_S;
 18:      6(* 3,1,2,1 *),     9(* 3,0,1,1 *)        = FAST (1-(1/2))*1*DELTA;
 19:      6(* 3,1,2,1 *),     8(* 3,1,1,0 *)        = FAST (1/2)*1*DELTA;
 20:      6(* 3,1,2,1 *),     1(* 3,2,2,1 DEATH  *) = (3-1)*LAMBDA_P;
 21:      6(* 3,1,2,1 *),    10(* 3,1,2,2 *)        = 2*LAMBDA_S;
 22:      7(* 3,0,2,2 *),    10(* 3,1,2,2 *)        = (3-0)*LAMBDA_P;
 23:      8(* 3,1,1,0 *),    11(* 3,0,0,0 *)        = FAST (1-(0/1))*1*DELTA;
 24:      8(* 3,1,1,0 *),     1(* 3,2,1,0 DEATH  *) = (3-1)*LAMBDA_P;
 25:      8(* 3,1,1,0 *),    12(* 3,1,1,1 *)        = 1*LAMBDA_S;
 26:      9(* 3,0,1,1 *),    12(* 3,1,1,1 *)        = (3-0)*LAMBDA_P;
 27:     10(* 3,1,2,2 *),    12(* 3,1,1,1 *)        = FAST (2/2)*1*DELTA;
 28:     10(* 3,1,2,2 *),     1(* 3,2,2,2 DEATH  *) = (3-1)*LAMBDA_P;
 29:     11(* 3,0,0,0 *),    13(* 3,1,0,0 *)        = (3-0)*LAMBDA_P;
 30:     12(* 3,1,1,1 *),    13(* 3,1,0,0 *)        = FAST (1/1)*1*DELTA;
 31:     12(* 3,1,1,1 *),     1(* 3,2,1,1 DEATH  *) = (3-1)*LAMBDA_P;
 32:     13(* 3,1,0,0 *),     1(* 3,2,0,0 DEATH  *) = (3-1)*LAMBDA_P;
 33:
 34: (* NUMBER OF STATES IN MODEL = 13 *)
 35: (* NUMBER OF TRANSITIONS IN MODEL = 24 *)
 36: (* 6 DEATH STATES AGGREGATED INTO STATE 1 *)

      0.02 SECS. TO READ MODEL FILE
 37? run

 MODEL FILE = triadcold.mod                SURE V7.9 22 Sep 97  14:20:50

                LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
  -----------   -----------   -----------   ---------------------------------
                1.66645e-10   1.69427e-10   <prune 1.2e-19>

 25 PATH(S) TO DEATH STATES 2 PATH(S) PRUNED
 HIGHEST PRUNE LEVEL =  2.03357e-17
 0.000 SECS. CPU TIME UTILIZED
```

The assist model (`triadcold.ast`) is:

```
(*  TRIAD WITH COLD SPARES  *)

N_PROCS = 3;            (* Number of active processors *)
```

```
N_SPARES = 2;           (* Number of spare processors *)
LAMBDA_P = 1E-4;        (* Failure rate of active processors *)
LAMBDA_S = 1E-5;        (* Failure rate of spare processors *)
DELTA = 3.6E3;          (* Reconfiguration rate *)

SPACE = (NP: 0..N_PROCS, (* Number of active processors *)
         NFP: 0..N_PROCS, (* Number of failed active processors *)
         NS: 0..N_SPARES, (* Number of spare processors *)
         NFS: 0..N_SPARES); (* Number of failed spare processors *)

START = (N_PROCS, 0, N_SPARES, 0);

DEATHIF 2 * NFP >= NP;

IF NP > NFP TRANTO NFP = NFP+1 BY (NP-NFP)*LAMBDA_P;
   (* Active processor failure *)

IF NS > NFS TRANTO NFS = NFS+1 BY NS*LAMBDA_S;
   (* Spare processor failure *)

IF (NFP > 0 AND NS > 0) THEN
   IF NS > NFS   (* Replace failed processor with working spare *)
      TRANTO (NP, NFP-1, NS-1, NFS)
             BY FAST (1-(NFS/NS))*NFP*DELTA;
   IF NFS > 0    (* Replace failed processor with failed spare *)
      TRANTO (NP, NFP, NS-1, NFS-1)
             BY FAST (NFS/NS)*NFP*DELTA;
ENDIF;
```

6.5 Example 4

This interactive session illustrates how SURE can be used to obtain system unreliability as a function of mission time.

```
$ sure

  SURE V5.2    NASA Langley Research Center

  1? read ftmp9

  2: LAMBDA = 5E-4;              (* PERMANENT FAULT RATE *)
  3: STDEV = 3.6E-4;             (* STAN. DEV. OF RECOVERY DISTRIBUTION *)
  4: RECOVER = 2.7E-4;           (* MEAN OF RECOVERY DISTRIBUTION *)
  5: TIME = 0.1 TO* 1000 BY 10;
  6: 1,2 = 9*LAMBDA;
  7: 2,3 = 2*LAMBDA;
  8: 2,4 = ;
  9: 4,5 = 9*LAMBDA;
```

```
10: 5,6 = 2*LAMBDA;
11: 5,7 = ;
12: 7,8 = 6*LAMBDA;
13: 8,9 = 2*LAMBDA;
14: 8,10 = ;
15: 10,11 = 6*LAMBDA;
16: 11,12 = 2*LAMBDA;
17: 11,13 = ;
18: 13,14 = 6*LAMBDA;
19: 14,15 = 2*LAMBDA;
20: 14,16 = ;
21: 16,17 = 3*LAMBDA;
22: 17,18 = 2*LAMBDA;
23: 17,19 = ;
24: 19,20 = 1*LAMBDA;
25: START = 1;

26? qtcalc = 0;                 (* use algebraic Q(T) calculation *)

27? run

MODEL FILE = ftmp9.mod                    SURE V7.9 22 Sep 97  13:50:57


   TIME        LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
 -----------   -----------   -----------   ---------------------------------
 1.00000e-01   1.01365e-10   1.21528e-10   <prune 7.6e-16>
 1.00000e+00   1.14931e-09   1.21774e-09   <prune 4.6e-15>
 1.00000e+01   1.19341e-08   1.24273e-08   <prune 1.1e-12>
 1.00000e+02   1.25980e-07   1.26748e-07   <ExpMat>
 1.00000e+03   7.68092e-03   7.69898e-03   <ExpMat>

7 PATH(S) TO DEATH STATES 1 PATH(S) PRUNED
HIGHEST PRUNE LEVEL =  2.43000e-12
Q(T) ACCURACY >= 11 DIGITS
0.017 SECS. CPU TIME UTILIZED
28? exit
```

6.6 Example 5

This example illustrates the use of SURE to solve a model of a triplex system with transient and permanent faults. The permanent faults arrive at rate `LAMBDA` and the transient faults arrive at rate `GAMMA`. In the presence of a single fault the system degrades to a simplex at rate `DELTA`. The operating system sometimes improperly degrades in the presence of a transient fault. This occurs at rate `PHI`. This model contains a loop and thus illustrates SURE's loop truncation method.

```
$ sure
```

```
 SURE V5.2   NASA Langley Research Center

 1? read 3trans

 2: LAMBDA = 1E-4;             (* FAULT ARRIVAL RATE *)
 3: INPUT DELTA;               (* RECOVERY RATE *)
    DELTA? 1800
 4: GAMMA = 10*LAMBDA;         (* TRANSIENT FAULT RATE *)
 5: RHO = 1 TO* 1E7 BY 10;     (* RATE OF DISAPPEARANCE OF TRANSIENT FAULTS *)
 6: PHI = DELTA;               (* RATE TRANSIENTS RECONFIGURED OUT *)
 7: T = RHO + DELTA;
 8:
 9: 1,2 = 3*LAMBDA;
10: 2,3 = 2*LAMBDA + 2*GAMMA;
11: 2,4 = ;
12: 4,5 = LAMBDA + GAMMA;
13: 1,6 = 3*GAMMA;
14: 6,1 = ;
15: 6,4 = ;
16: 6,7 = 2*LAMBDA + 2*GAMMA;

      0.00 SECS. TO READ MODEL FILE
18? run

MODEL FILE = 3trans.mod                    SURE V7.9 22 Sep 97  13:52:40

*** START STATE ASSUMED TO BE 1

DELTA =  1.800e+03,

   RHO          LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
 -----------   -----------   -----------   ----------------------------------
 1.00000e+00   1.77763e-04   1.81450e-04   <prune 1.4e-13>
 1.00000e+01   1.76971e-04   1.80638e-04   <prune 6.3e-15>
 1.00000e+02   1.69461e-04   1.72945e-04   <prune 5.4e-12>
 1.00000e+03   1.20686e-04   1.23039e-04   <prune 1.7e-09>
 1.00000e+04   4.12797e-05   4.20395e-05   <prune 5.3e-09>
 1.00000e+05   1.92296e-05   1.96180e-05   <prune 4.0e-09>
 1.00000e+06   1.66249e-05   1.69695e-05   <prune 2.8e-10>
 1.00000e+07   1.63596e-05   1.67010e-05   <prune 1.0e-09>

14 PATH(S) TO DEATH STATES 4 PATH(S) PRUNED
HIGHEST PRUNE LEVEL =  3.33002e-09
0.067 SECS. CPU TIME UTILIZED
```

6.7 Example 6

This example illustrates the use of SURE in Lee mode. The same model as used in example 5 is used here. However, the information given for the fast recovery transitions is different. In the presence of a permanent fault, the system degrades to a simplex. The mean degradation time is `1/DELTA`. The probability that the degradation process takes more than `QUANT2` hours is `QPROB2.` In the presence of a transient fault, the system degrades to a simplex with probability `PHI/(PHI+RHO)` and returns to the fault-free state with probability `RHO/(RHO+PHI)`. The probability that this requires more than `QUANT6` hours is `QPROB6`.

```
$ sure

  SURE V5.2    NASA Langley Research Center

  1? read leem

  2: LEE;
  3: LAMBDA = 1E-4;             (* FAULT ARRIVAL RATE *)
  4: DELTA = 1800.0;            (* MEAN RECOVERY TIME *0
  5: GAMMA = 10*LAMBDA;         (* TRANSIENT FAULT RATE *)
  6: RHO = 1 TO* 1E7 BY 10;     (* RECOVERY RATE FROM TRANSIENT FAULT *)
  7: PHI = DELTA;               (* RATE TRANSIENTS RECONFIGURED OUT *)
  8: T = RHO + PHI;
  9: QUANT2 = 1E-2;
 10: QPROB2 = 1.0 - EXP(-DELTA*QUANT2);
 11: TIME = 10;
 12: 1,2 = 3*LAMBDA;
 13: 2,3 = 2*LAMBDA + 2*GAMMA;
 14: @2 = ;
 15: 2,4 = ;
 16: 4,5 = LAMBDA + GAMMA;
 17: 1,6 = 3*GAMMA;
 18: QUANT6 = 1E-2;
 19: QPROB6 = 1.0 - EXP(-T*QUANT6);
 20: @6 = ;
 21: 6,1 = ;
 22: 6,4 = ;
 23: 6,7 = 2*LAMBDA + 2*GAMMA;

 24? run

 *** START STATE ASSUMED TO BE 1

 ----- LEE STATISTICAL ANALYSIS MODE -----

   RHO           LOWERBOUND    UPPERBOUND    COMMENTS    RUN #1
  -----------   -----------   -----------   ---------------------
  1.00000E+00   1.78430E-04   1.81450E-04
 1.00000E+01    1.77632E-04   1.80639E-04
 1.00000E+02    1.70066E-04   1.72945E-04
 1.00000E+03    1.20986E-04   1.23038E-04
 1.00000E+04    4.13316E-05   4.20343E-05
 1.00000E+05    1.92859E-05   1.96141E-05
 1.00000E+06    1.66853E-05   1.69692E-05
 1.00000E+07    1.64206E-05   1.67000E-05

12 PATH(S) PROCESSED
1 LOOP(S) TRUNCATED AT DEPTH 3
3.750 SECS. CPU TIME UTILIZED
```

6.8 Example 7

This example illustrates the use of Lee's method to model a system with two possible recoveries from a fault. In this model, the system recovers from a fault by bringing in a (nonfailed) spare 90 percent of the time and degrades to a simplex 10 percent of the time.

```
$ sure

  SURE V5.2    NASA Langley Research Center

  1? lee;

  2? lambda = 1e-4;       (* Failure rate of a processor *)
  3? pr1 = 0.90;          (* Probability recovery is by sparing *)
  4? mu = 2e-4;           (* Mean recovery time *)
  5? 1,2 = 3*lambda;
  6? 2,3 = 2*lambda;
  7? 2,4 = ;
  8? 4,5 = 3*lambda;
  9? 5,6 = 2*lambda;
 10? 2,7 = ;
 11? 7,8 = lambda;
 12? @2 = ;   (* No observed recoveries greater than 2*MU*)
 13? list=2;
 14? run

 MODEL FILE = leem.mod                      SURE V7.9 22 Sep 97  13:55:13


 *** START STATE ASSUMED TO BE 1
 ----- LEE STATISTICAL ANALYSIS MODE -----


    RHO          LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
  -----------   -----------   -----------   ---------------------------------
  1.00000e+00   1.78430e-04   1.81450e-04   <prune 1.5e-12>
  1.00000e+01   1.77632e-04   1.80639e-04    <prune 1.4e-10>
  1.00000e+02   1.70066e-04   1.72958e-04    <prune 1.2e-08>
  1.00000e+03   1.20986e-04   1.23039e-04    <prune 1.7e-09>
  1.00000e+04   4.13316e-05   4.20396e-05    <prune 5.3e-09>
  1.00000e+05   1.92859e-05   1.96180e-05    <prune 4.0e-09>
  1.00000e+06   1.66853e-05   1.69729e-05    <prune 3.6e-09>
  1.00000e+07   1.64206e-05   1.67010e-05    <prune 1.0e-09>

 14 PATH(S) TO DEATH STATES 1 PATH(S) PRUNED
 HIGHEST PRUNE LEVEL =  3.45891e-08
 0.067 SECS. CPU TIME UTILIZED
```

6.9 Example 8

The following session illustrates the use of the `ORPROB` command:

```
$SURE

  SURE V7.5     NASA Langley Research Center

  1? 1,2 = 1E-4;

  2? RUN

                 LOWERBOUND    UPPERBOUND    COMMENTS    RUN #1

  ------------   -----------   -----------   ---------------------

                9.99500E-04   1.00000E-03

  1 PATH(S) PROCESSED
  0.070 SECS. CPU TIME UTILIZED

  3? 2,4 = 1E-5;

  4? RUN

                 LOWERBOUND    UPPERBOUND    COMMENTS    RUN #2
  ------------   -----------   -----------   ---------------------
                9.99500E-05   1.00000E-04

 1 PATH(S) PROCESSED
 0.050 SECS. CPU TIME UTILIZED

 5? 1,2 = 2.5E-4;

 6? RUN
```

```
                LOWERBOUND    UPPERBOUND    COMMENTS    RUN #3
 ------------   -----------   -----------   ---------------------
               2.49687E-03   2.50000E-03

1 PATH(S) PROCESSED
0.040 SECS. CPU TIME UTILIZED

7? ORPROB

   RUN #      LOWERBOUND     UPPERBOUND
----------   -----------    -----------
    1        9.99500E-04    1.00000E-03
    2        9.99500E-05    1.00000E-04
    3        2.49687E-03    2.50000E-03
----------   -----------    -----------

 OR PROB =   3.59352E-03    3.59715E-03

8? EXIT
```

6.10 Example 9

In this example a model of a triad with spares is investigated. When an active processor fails, a spare processor is brought into the configuration to replace the faulty one. If a spare fails, the fault remains undetectable until it is brought into the active configuration. For simplicity the time required to replace a faulty processor with a spare and the degradation time are assumed to be exponentially distributed. Therefore, the `FAST` exponential specification method can be used:

```
$ sure

  SURE V7.5    NASA Langley Research Center

  1? read undet

  2: LAMBDA = 1E-4;                 (* Failure rate of a processor *)
  3: DELTA = 1E4;                   (* Rate of sparing *)
  4: DEGRATE = 1E4;                 (* Rate of degrading to a simplex *)
  5: PSI = 1E-6 TO* LAMBDA BY 10;   (* Failure rate of spares *)
  6: 1,2 = 3*LAMBDA;
  7: 2,3 = 2*LAMBDA;
  8: 1,7 = PSI;
  9: 2,4 = FAST DELTA;
 10: 2,8 = PSI;
 11: 4,5 = 3*LAMBDA;
 12: 5,6 = 2*LAMBDA;
 13: 5,10 = FAST DEGRATE;
 14: 7,8 = 3*LAMBDA;
```

```
15: 8,9 = 2*LAMBDA;
16: 8,5 = FAST DELTA;
17: 10,11 = LAMBDA;
18? RUN

MODEL FILE = undet.mod                    SURE V7.9 22 Sep 97  13:56:43


   PSI          LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
 -----------   -----------   -----------    --------------------------------
 1.00000e-06   1.55410e-09   1.56509e-09
 1.00000e-05   1.59876e-09   1.61010e-09
 1.00000e-04   2.04524e-09   2.06016e-09

9 PATH(S) TO DEATH STATES
0.016 SECS. CPU TIME UTILIZED
```

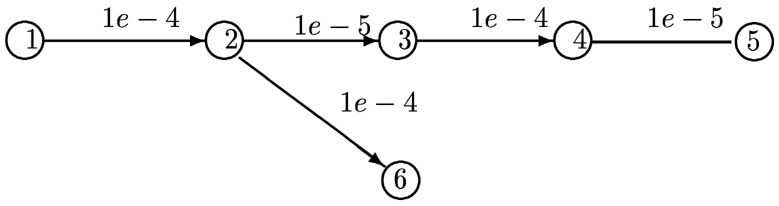
6.11 Example 10

This example illustrates the use of the IF command to analyze the probability of system failure of a N-multiply redundant (NMR) system as a function of N:

```
$ SURE

  SURE V7.5    NASA Langley Research Center

  1? read nmr

  2: LAMBDA = 1E-4;
  3: N = 3 TO 15 BY 2;
  4: 1,2 = N*LAMBDA;
  5: IF N > 2 THEN 2,3 = (N-1)*LAMBDA;
  6: IF N > 4 THEN 3,4 = (N-2)*LAMBDA;
  7: IF N > 6 THEN 4,5 = (N-3)*LAMBDA;
  8: IF N > 8 THEN 5,6 = (N-4)*LAMBDA;
  9: IF N > 10 THEN 6,7 = (N-5)*LAMBDA;
 10: IF N > 12 THEN 7,8 = (N-6)*LAMBDA;
 11: IF N > 14 THEN 8,9 = (N-7)*LAMBDA;

 12? run

 MODEL FILE = nmr.mod                     SURE V7.9 22 Sep 97  13:57:23


    N           LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
  -----------   -----------   -----------    --------------------------------
  3.00000e+00   2.99500e-06   3.00000e-06
  5.00000e+00   9.97000e-09   9.99999e-09
  7.00000e+00   3.48460e-11   3.50000e-11
  9.00000e+00   1.25265e-13   1.26000e-13
  1.10000e+01   4.58634e-16   4.62000e-16
  1.30000e+01   1.70099e-18   1.71600e-18
  1.50000e+01   6.36922e-21   6.43500e-21

 1 PATH(S) TO DEATH STATES
 0.050 SECS. CPU TIME UTILIZED
```

6.12 Example 11

In this example the use of the PRUNE constant is demonstrated. Consider the following model



The probability of reaching state 5 is very small in comparison with state 6. By specifying a prune level of 5e-11, SURE will prune the path to state 5:

```
% sure

  SURE V7.5   NASA Langley Research Center

  1? read prune_ex

  2: 1,2 = 1e-4;
  3: 2,3 = 1e-5;
  4: 2,6 = 1e-4;
  5: 3,4 = 1e-4;
  6: 4,5 = 1e-5;
  7:
  8: list=2;

  9? prune = 5e-11;
 10? run

 MODEL FILE = prune_ex.mod                SURE V7.5 16 Aug 90   10:06:15


 DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
 ----------   -----------   -----------    ---------------------------------
      6       4.99650e-07   5.00000e-07
 sure prune   0.00000e+00   1.66667e-11
 TOTAL        4.99650e-07   5.00017e-07

  1 PATH(S) TO DEATH STATES, 1 PATH(S) PRUNED AT LEVEL  5.00000e-11
```

Note that the probability of the path upto the pruning point is included in the upper bound.
If no `PRUNE` level is set, the SURE program will automatically determine a pruing level:

```
% sure

  SURE V7.5   NASA Langley Research Center

  1? read prune_ex

  2: 1,2 = 1e-4;
  3: 2,3 = 1e-5;
  4: 2,6 = 1e-4;
  5: 3,4 = 1e-4;
  6: 4,5 = 1e-5;
  7:
  8: list=2;

  9? run

 MODEL FILE = prune_ex.mod                 SURE V7.5 16 Aug 90   10:02:42


 DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                RUN #1
 ----------   -----------   -----------    ---------------------------------
      6       4.99650e-07   5.00000e-07
 sure prune   0.00000e+00   1.66667e-11

 TOTAL        4.99650e-07   5.00017e-07

  1 PATH(S) TO DEATH STATES, 1 PATH(S) PRUNED
  HIGHEST PRUNE LEVEL =  1.00000e-10
```

This is the default mode of the SURE program. The SURE program selects a prune level based on the probability of the first death state it encounters. As more death states are encountered, the program updates the value of `PRUNE`. The highest level of pruning is reported with the message:

```
 HIGHEST PRUNE LEVEL = x.xxxE-xx
```

In this example the highest prune level was `1.00000e-10`.
To turn off autopruning, the USER must enter:

```
    AUTOPRUNE = 0;
```

before the `RUN` command.

A Appendix A — Phased Missions

A.1 Phased Missions

Many systems exhibit different failure behaviors or operational characteristics during different phases of a mission. For example, a spacecraft may experience considerably higher component failure rates during liftoff than in the weightless, benign environment of space. Also, the failure of a particular component may be catastophic only during a specific phase, such as the three-minute landing phase of an aircraft.

In a phased-mission solution, a model is solved for the first phase of the mission. The final probabilities of the operational states are used to calculate the initial state probabilities for a second model. (The second model usually differs from the first model in some manner.) This process is repeated for as many phases as there are in the mission.

The SURE program reports upper and lower bounds on the operational states just as for the death states. These bounds are not as tight as the death state probabilities, but are usually acceptable. The upper and lower bounds on recovery states (i.e. states with fast transitions leaving them) are usually quite crude. Fortunately, these states usually have operational probabilities which are several orders of magnitude lower than the other operational states in the model because systems typically spend a very small percentage of their operational time performing recoveries. Thus, the crudeness of the bounds for the recovery states has a negligible impact on the accuracy of phased mission calculations when these values are used as initial probabilities in subsequent phases.

Suppose we have a system which operates in two basic phases—(1) cruise and (2) landing. The system is implemented using a triad of processors and two warm spares. For simplicity, we will assume perfect detection of spare failure. During the cruise phase which lasts for 2 hours, the system reconfigures by sparing and degradation. After the cruise phase, the system goes into a landing phase which lasts 3 minutes. During this phase, the workload on the machines is so high that the additional processing that would be needed to perform reconfiguration cannot be tolerated. Therefore, the system is designed to "turn off" the reconfiguration processes during this phase.

In order to model this two-phased mission, two different models must be created—one for each phase. The following ASSIST input describes a model for the cruise phase:

```
NSI = 2;                        (* Number of spares initially *)
LAMBDA = 1E-4;                  (* Failure rate of active processors *)
GAMMA = 1E-6;                   (* Failure rate of spares *)
TIME = 2.0;                     (* Mission time *)

MU = 7.9E-5;                    (* Mean time to replace with spare *)
SIGMA = 2.56E-5;                (* Stan. dev. of time to replace with spare *)

MU_DEG = 6.3E-5;                (* Mean time to degrade to simplex *)
SIGMA_DEG = 1.74E-5;            (* Stan. dev. of time to degrade to simplex *)

SPACE = (NW: 0..3,              (* Number of working processors *)
         NF: 0..3,              (* Number of failed active procssors *)
         NS: 0..NSI);           (* Number of spares *)

START = (3,0,NSI);
```

```
LIST=3;

IF NW > 0                               (* A processor can fail *)
   TRANTO (NW-1,NF+1,NS) BY NW*LAMBDA;

IF (NF > 0) AND (NS > 0)                (* A spare becomes active *)
   TRANTO (NW+1,NF-1,NS-1) BY <MU,SIGMA>;

IF (NF > 0) AND (NS = 0)                (* No more spares, degrade to simplex *)
   TRANTO (1,0,0) BY <MU_DEG,SIGMA_DEG>;

IF NS > 0                               (* A spare fails and is detected *)
   TRANTO (NW,NF,NS-1) BY NS*GAMMA;

DEATHIF NF >= NW;
```

The ASSIST program generates the following SURE model.

```
NSI = 2;
LAMBDA = 1E-4;
GAMMA = 1E-6;
TIME = 2.0;
MU = 7.9E-5;
SIGMA = 2.56E-5;
MU_DEG = 6.3E-5;
SIGMA_DE = 1.74E-5;
LIST = 3;


    2(* 3,0,2 *),    3(* 2,1,2 *) = 3*LAMBDA;
    2(* 3,0,2 *),    4(* 3,0,1 *) = 2*GAMMA;
    3(* 2,1,2 *),    1(* 1,2,2 *) = 2*LAMBDA;
    3(* 2,1,2 *),    4(* 3,0,1 *) = <MU,SIGMA>;
    3(* 2,1,2 *),    5(* 2,1,1 *) = 2*GAMMA;
    4(* 3,0,1 *),    5(* 2,1,1 *) = 3*LAMBDA;
    4(* 3,0,1 *),    6(* 3,0,0 *) = 1*GAMMA;
    5(* 2,1,1 *),    1(* 1,2,1 *) = 2*LAMBDA;
    5(* 2,1,1 *),    6(* 3,0,0 *) = <MU,SIGMA>;
    5(* 2,1,1 *),    7(* 2,1,0 *) = 1*GAMMA;
    6(* 3,0,0 *),    7(* 2,1,0 *) = 3*LAMBDA;
    7(* 2,1,0 *),    1(* 1,2,0 *) = 2*LAMBDA;
    7(* 2,1,0 *),    8(* 1,0,0 *) = <MU_DEG,SIGMA_DEG>;
    8(* 1,0,0 *),    1(* 0,1,0 *) = 1*LAMBDA;

(* NUMBER OF STATES IN MODEL      = 8 *)
(* NUMBER OF TRANSITIONS IN MODEL = 14 *)
(* 4 DEATH STATES AGGREGATED STATES 1 - 1 *)
```

The model for the second phase (call it "phaz2.mod") is easily created with an editor by deleting the reconfiguration transitions and changing the mission time to 0.05 hours. The resulting file is:

```
NSI = 2;
LAMBDA = 1E-4;
GAMMA = 1E-6;
TIME = 0.05;
LIST = 3;

    2(* 3,0,2 *),    3(* 2,1,2 *) = 3*LAMBDA;
    2(* 3,0,2 *),    4(* 3,0,1 *) = 2*GAMMA;
    3(* 2,1,2 *),    1(* 1,2,2 *) = 2*LAMBDA;

    3(* 2,1,2 *),    5(* 2,1,1 *) = 2*GAMMA;
    4(* 3,0,1 *),    5(* 2,1,1 *) = 3*LAMBDA;
    4(* 3,0,1 *),    6(* 3,0,0 *) = 1*GAMMA;
    5(* 2,1,1 *),    1(* 1,2,1 *) = 2*LAMBDA;

    5(* 2,1,1 *),    7(* 2,1,0 *) = 1*GAMMA;
    6(* 3,0,0 *),    7(* 2,1,0 *) = 3*LAMBDA;
    7(* 2,1,0 *),    1(* 1,2,0 *) = 2*LAMBDA;

    8(* 1,0,0 *),    1(* 0,1,0 *) = 1*LAMBDA;
```

The SURE program is the executed on the first model (stored in file "phaz.mod"), using the LIST = 3 option. This causes the SURE program to output all of the operational state probabities as well as the death state probabilities. This is illustrated below:

```
  SURE V7.2   NASA Langley Research Center

  1? read0 phaz

 31? run

 MODEL FILE = phaz.mod                    SURE V7.2 11 Jan 90   13:56:49


 DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
 ----------   -----------   -----------    ---------------------------------
      1       9.35692e-12   9.48468e-12

 TOTAL        9.35692e-12   9.48468e-12


 OPER-STATE    LOWERBOUND    UPPERBOUND
 ----------   -----------   -----------
```

```
     2       9.99396e-01   9.99396e-01
     3       0.00000e+00   1.53952e-06
     4       6.02277e-04   6.03819e-04
     5       0.00000e+00   1.43291e-09
     6       1.80332e-07   1.81768e-07
     7       0.00000e+00   5.59545e-13
     8       3.57995e-11   3.63591e-11


 20 PATH(S) PROCESSED
 0.617 SECS. CPU TIME UTILIZED
 32? exit
```

The SURE program also creates a file containing these probabilities in a format that can be used to initialize the states for the next phase. The SURE program names the file “phaz.ini”, i.e. adds “.ini” to the file name. The contents of this file generated by the run above is:

```
INITIAL_PROBS(
    1: ( 9.35692e-12, 9.48468e-12),
    2: ( 9.99396e-01, 9.99396e-01),
    3: ( 0.00000e+00, 1.53952e-06),
    4: ( 6.02277e-04, 6.03819e-04),
    5: ( 0.00000e+00, 1.43291e-09),
    6: ( 1.80332e-07, 1.81768e-07),
    7: ( 0.00000e+00, 5.59545e-13),
    8: ( 3.57995e-11, 3.63591e-11)
  );
```

Next, the SURE program is executed on the second model. The state probabilities are initialized using the SURE `INITIAL_PROBS` command. Note that “.ini” file output is in the correct format for the SURE program:

```
air51% sure

  SURE V7.2   NASA Langley Research Center

  1? read0 phaz2

 31? read phaz.ini

 32: INITIAL_PROBS(
 33:     1: ( 9.35692e-12, 9.48468e-12),
 34:     2: ( 9.99396e-01, 9.99396e-01),
 35:     3: ( 0.00000e+00, 1.53952e-06),
 36:     4: ( 6.02277e-04, 6.03819e-04),
 37:     5: ( 0.00000e+00, 1.43291e-09),
 38:     6: ( 1.80332e-07, 1.81768e-07),
 39:     7: ( 0.00000e+00, 5.59545e-13),
40:     8: ( 3.57995e-11, 3.63591e-11)
41:   );

42? run

MODEL FILE = phaz.ini                     SURE V7.2 11 Jan 90   13:58:12


DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
----------   -----------   -----------    ---------------------------------
     1       8.43564e-11   9.98944e-11

TOTAL        8.43564e-11   9.98944e-11


OPER-STATE    LOWERBOUND    UPPERBOUND
----------   -----------   -----------
     2       9.99381e-01   9.99381e-01
     3       1.49908e-05   1.65304e-05
     4       6.02368e-04   6.03910e-04
     5       9.03554e-09   1.04918e-08
     6       1.80359e-07   1.81795e-07
     7       2.70540e-12   3.28658e-12
     8       3.57993e-11   3.63589e-11


  9 PATH(S) PRUNED AT LEVEL  1.49540e-16
  SUM OF PRUNED STATES PROBABILITY <  5.04017e-18

9 PATH(S) PROCESSED
0.417 SECS. CPU TIME UTILIZED
43?
```

A.2 Non-Constant Failure Rates

In the previous section, a two-phased system was analyzed which required different models for each of the phases. A related situation occurs when the structure of the model remains the same, but some parameters, such as the failure rates, change from one phase to another.

Consider a triad with warm spares that experiences different failure rates for each of the phases:

- phase 1 (6 min): $\lambda = 2 \times 10^{-4}$, $\gamma = 10^{-4}$
- phase 2 (2 hours): $\lambda = 10^{-4}$, $\gamma = 10^{-5}$
- phase 3 (3 min): $\lambda = 10^{-3}$, $\gamma = 10^{-4}$

The same SURE model can be used for all of the phases, and the user can be prompted for the parameter values using the SURE `INPUT` command:

```
INPUT LAMBDA, GAMMA, TIME;
```

The full SURE model, stored in file “`phase.mod`,”is:

```
INPUT LAMBDA, GAMMA, TIME;
NSI = 2;
MU = 7.9E-5;
SIGMA = 2.56E-5;
MU_DEG = 6.3E-5;
SIGMA_DE = 1.74E-5;
LIST = 3;
QTCALC = 1;


    2(* 3,0,2 *),    3(* 2,1,2 *) = 3*LAMBDA;
    2(* 3,0,2 *),    4(* 3,0,1 *) = 2*GAMMA;
    3(* 2,1,2 *),    1(* 1,2,2 *) = 2*LAMBDA;
    3(* 2,1,2 *),    4(* 3,0,1 *) = <MU,SIGMA>;
    3(* 2,1,2 *),    5(* 2,1,1 *) = 2*GAMMA;
    4(* 3,0,1 *),    5(* 2,1,1 *) = 3*LAMBDA;
    4(* 3,0,1 *),    6(* 3,0,0 *) = 1*GAMMA;
    5(* 2,1,1 *),    1(* 1,2,1 *) = 2*LAMBDA;
    5(* 2,1,1 *),    6(* 3,0,0 *) = <MU,SIGMA>;
    5(* 2,1,1 *),    7(* 2,1,0 *) = 1*GAMMA;
    6(* 3,0,0 *),    7(* 2,1,0 *) = 3*LAMBDA;
    7(* 2,1,0 *),    1(* 1,2,0 *) = 2*LAMBDA;
    7(* 2,1,0 *),    8(* 1,0,0 *) = <MU_DEG,SIGMA_DEG>;
    8(* 1,0,0 *),    1(* 0,1,0 *) = 1*LAMBDA;
```

The `QTCALC = 1` command causes the SURE program to use more accurate (but slower) numerical routines. This increased accuracy is often necessary when analyzing phased missions. The interactive session follows:

```
  SURE V7.2   NASA Langley Research Center

  1? read0 phase

     LAMBDA? 2e-4

     GAMMA? 1e-4

     TIME? .1

 30? run

 MODEL FILE = phase.mod                  SURE V7.2 12 Jan 90   09:35:50


 TIME =  1.000e-01,  GAMMA =  1.000e-04,  LAMBDA =  2.000e-04,
DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #1
----------    -----------   -----------   ---------------------------------
     1        1.78562e-12   1.89600e-12   <ExpMat>

TOTAL         1.78562e-12   1.89600e-12   <ExpMat - 14,14>


OPER-STATE    LOWERBOUND    UPPERBOUND
----------    -----------   -----------
     2        9.99920e-01   9.99920e-01   <ExpMat>
     3        0.00000e+00   9.98043e-07   <ExpMat>
     4        7.89960e-05   7.99941e-05   <ExpMat>
     5        0.00000e+00   1.14966e-10   <ExpMat>
     6        2.67751e-09   2.80076e-09   <ExpMat>
     7        0.00000e+00   5.17358e-15   <ExpMat>
     8        5.08706e-14   5.60442e-14   <ExpMat>


 10 PATH(S) PRUNED AT LEVEL  4.75740e-20
 SUM OF PRUNED STATES PROBABILITY <  6.11113e-20
 Q(T) ACCURACY >= 14 DIGITS

10 PATH(S) PROCESSED
2.867 SECS. CPU TIME UTILIZED
31? read0 phase

    LAMBDA? 1e-4

    GAMMA? 1e-5

    TIME? 2.0

60? read phase.ini

61: INITIAL_PROBS(
62:     1: ( 1.78562e-12, 1.89600e-12),
63:     2: ( 9.99920e-01, 9.99920e-01),
64:     3: ( 0.00000e+00, 9.98043e-07),
65:     4: ( 7.89960e-05, 7.99941e-05),
66:     5: ( 0.00000e+00, 1.14966e-10),
67:     6: ( 2.67751e-09, 2.80076e-09),
68:     7: ( 0.00000e+00, 5.17358e-15),
69:     8: ( 5.08706e-14, 5.60442e-14)
70:   );
```

```
      0.07 SECS. TO READ MODEL FILE
71? run

MODEL FILE = phase.ini                   SURE V7.2 12 Jan 90   09:36:19


TIME =  2.000e+00,  GAMMA =  1.000e-05,  LAMBDA =  1.000e-04,


DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #2
----------   -----------   -----------    ---------------------------------
     1       1.11438e-11   1.13950e-11    <ExpMat>

TOTAL        1.11438e-11   1.13950e-11    <ExpMat - 14,14>


OPER-STATE    LOWERBOUND    UPPERBOUND
----------   -----------   -----------
     2       9.99280e-01   9.99280e-01    <ExpMat>
     3       0.00000e+00   2.35621e-06    <ExpMat>
     4       7.17134e-04   7.20490e-04    <ExpMat>
     5       0.00000e+00   2.82024e-09    <ExpMat>
     6       2.48355e-07   2.51362e-07    <ExpMat>
     7       0.00000e+00   1.19806e-12    <ExpMat>
     8       5.53210e-11   5.65243e-11    <ExpMat>


 30 PATH(S) PRUNED AT LEVEL  4.61326e-19
 SUM OF PRUNED STATES PROBABILITY <  1.15985e-18
 Q(T) ACCURACY >= 14 DIGITS

19 PATH(S) PROCESSED
4.267 SECS. CPU TIME UTILIZED
72? read0 phase

    LAMBDA? 1e-3

    GAMMA? 1e-4

    TIME? 0.05

101? read phase.ini

102: INITIAL_PROBS(
103:     1: ( 1.11438e-11, 1.13950e-11),
104:     2: ( 9.99280e-01, 9.99280e-01),
105:     3: ( 0.00000e+00, 2.35621e-06),
106:     4: ( 7.17134e-04, 7.20490e-04),
107:     5: ( 0.00000e+00, 2.82024e-09),
108:     6: ( 2.48355e-07, 2.51362e-07),
109:     7: ( 0.00000e+00, 1.19806e-12),
110:     8: ( 5.53210e-11, 5.65243e-11)
111:   );

112? run

 MODEL FILE = phase.ini                  SURE V7.2 12 Jan 90   09:36:57

 TIME =  5.000e-02,  GAMMA =  1.000e-04,  LAMBDA =  1.000e-03,


 DEATHSTATE    LOWERBOUND    UPPERBOUND    COMMENTS                 RUN #3
 ----------    -----------   -----------   ---------------------------------
      1        3.29083e-11   3.54718e-11   <ExpMat>

 TOTAL         3.29083e-11   3.54718e-11   <ExpMat - 14,14>


 OPER-STATE    LOWERBOUND    UPPERBOUND
 ----------    -----------   -----------
      2        9.99120e-01   9.99120e-01   <ExpMat>
      3        0.00000e+00   6.30518e-06   <ExpMat>
      4        8.72933e-04   8.82595e-04   <ExpMat>
      5        0.00000e+00   7.50836e-09   <ExpMat>
      6        3.68000e-07   3.78561e-07   <ExpMat>
      7        0.00000e+00   3.72751e-12   <ExpMat>
      8        9.99350e-11   1.04866e-10   <ExpMat>


  33 PATH(S) PRUNED AT LEVEL  8.23385e-18
  SUM OF PRUNED STATES PROBABILITY <  3.35190e-17
  Q(T) ACCURACY >= 14 DIGITS

 13 PATH(S) PROCESSED
 3.350 SECS. CPU TIME UTILIZED
113? exit
```

As in the previous section, the results of each previous phase are loaded by reading the ".ini" file created by the previous run. The `<ExpMat>` output in the `COMMENTS` field indicates that the more accurate 'verb'QTCALC=1' numerical routines were utilized.

A.3 Continuously Varying Failure Rates

Suppose that the failure rates change continuously in time as shown in figure 1. This type of failure rate is called a "decreasing failure rate". The SURE program cannot handle this type of failure

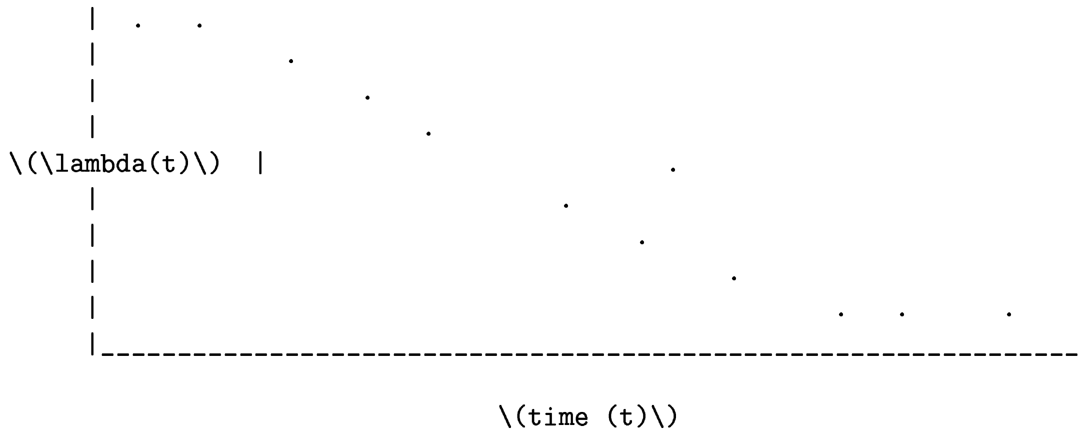


Figure 1: Decreasing Failure Rate Function

rate directly since it leads to "non-homogeneous" or "non-stationary" Markov models. However, good results can be obtained by using the phased-mission approach on a "linearized" upper bound shown in figure 2:

This problem requires nine steps, but is quite easy with the use of the ".ini" files. Because an upper bound is used for the failure rate, the result will be conservative. The problem can then be solved again using a consistently lower bound for the failure rate function to obtain a lower bound on the system failure probability.

B Appendix B: Error Messages

The following error messages are generated by the SURE system. These are listed in alphabetical order:
`ARGUMENT TO EXP FUNCTION MUST BE < 8.803E+01` — function is too large.
`ARGUMENT TO LN OR SQRT MUST BE > 0` — The LN and SQRT function require positive arguments.
`ARGUMENT TO STANDARD FUNCTION MISSING` — No argument was supplied for a standard function.
`COMMA EXPECTED` — Syntax error; a comma is needed.
`CONSTANT EXPECTED` — Syntax error; a constant is expected.
`DELTA > TIME` — the value of D used in the lower bound (i.e. Q(T-D)) is larger than the mission time. This can lead to a very poor lower bound. This is usually caused by using the fast-transition specification method to describe a slow transition, (i.e., a very slow recovery transition).
`DIVISION BY ZERO NOT ALLOWED` — A division by 0 was encountered when evaluating the expression.
`ERROR OPENING FILE` — The SURE system was unable to open the indicated file.
`Q(T) INACCURATE` — The entered mission time is too large for the default value of QTCALC. Therefore, the upper and lower bounds are very far apart. Set QTCALC equal to 1.
`Q(T) ~ x DIGITS` — The matrix exponential algorithm cannot guarantee more than x digits accuracy in the Q(T) calcu- lation.
`FILE NAME TOO LONG` — File names must be 80 or less characters.
`FILE NAME EXPECTED` — Syntax error, the file name is missing.

```
        | __________
        |  .   .   |____
        |             . |____
        |                   . |____
        |                       . |_____
\(\lambda(t)\) |                                      . |____
        |                               . |_____
        |                                     . |_______
        |                                            .  |_______________
        |                                                    .    .       .
        |_________________________________________________________________

                                             \(time (t)\)
```

Figure 2: Upper bound On Failure-Rate Function

`"id" CHANGED TO x` — The value of the identifier id is being changed to x.
`"id" CHANGED TO X TO Y` — The range of the variable "id" is being changed.
`"id" NOT FOUND` — The system is unable to SHOW the identifier since it has not yet been defined.
`IDENTIFIER EXPECTED` — Syntax error, identifier expected here.
`IDENTIFIER NOT DEFINED` — The identifier entered has not yet been defined.
`ILLEGAL CHARACTER` — The character used is not recognized by SURE.
`ILLEGAL INPUT VALUE` — A non-numeric character was entered in response to the INPUT command prompt.
`ILLEGAL STATEMENT` — The command word is unknown by the system.
`INPUT ALREADY DEFINED AS THE VARIABLE` — An attempt was made to input a value for an identifier that was already defined as the variable.
`INPUT LINE TOO LONG` — The command line exceeds the 100 character limit.
`INTEGER EXPECTED` — Syntax error, an integer is expected.
`LEE @ REQUIRES THREE PARAMETERS` — The @ statement requires three parameters in the LEE mode.
`MORE THAN ONE SOURCE STATE IN MODEL` — The model entered by the user has more than one source state (i.e., a state with no transitions into it). If a start state has been specified by a START command, it is used. Otherwise, the program arbitrarily chooses a start state.
`MUST BE IN "READ" MODE` — The INPUT command can be used only in a file processed by a READ command.
`NO RUNS MADE YET` — The ORPROB command was called before any runs were made.
`NUMBER TOO LONG` — Only 15 digits/characters allowed per number.
`ONLY 1 VARIABLE ALLOWED` — Only one variable can be defined per model.
`ONLY 100 VARIABLE RESULTS STORED` — the ORPROB command can only process the first 100 values of the variable per run.
`PRUNING TOO SEVERE` — the specified level of pruning is too large to guarantee that the bounds have WARNDIG digits of accuracy.
`RATE TOO FAST` — The upper and lower bounds are valid, but, an exponential transition in the model is too fast to permit close upper and lower bounds.

`REAL EXPECTED` — A floating point number is expected here.
`RECOVERY TOO SLOW` — The upper and lower bounds are valid, but, a nonexponential transition in the model is too slow to permit close upper and lower bounds.
`SEMICOLON EXPECTED` — Syntax error; a semicolon is needed.
`START STATE ASSUMED TO BE x` — There was no source state in the model and no start state was specified via a START command so the program arbitrarily selected x as the start state.
`ST. DEV TOO BIG` — The standard deviation of a fast distribution is too large to permit close upper and lower bounds; however, the bounds are valid.
`SUB-EXPRESSION TOO LARGE, i.e. > 1.70000E+38` — An overflow condition was encountered when evaluating the expression.
`THIS CONSTRUCT NOT PERMITTED IN LEE MODE` — This construct is not allowed while in the LEE mode.
`THIS CONSTRUCT NOT PERMITTED IN WHITE MODE` — This construct is not allowed while in the WHITE mode.
`TRANSITION NOT FOUND` — The system is unable to SHOW the transition because it has not yet been defined.
`TRUNC TOO SMALL` — The value of TRUNC is probably not large enough to guarantee that the upper bound is valid for this model. The user should rerun the model with a higher value of TRUNC.
`VMS FILE NOT FOUND` — The file indicated on the READ command is not present on the disk. (Note: make sure your default directory is correct.)
`0 STATES IN MODEL` — The RUN command found no states in the model.
`*** ERROR: HOLDING TIME AT x NOT DEFINED` — The holding time information (required in LEE mode) for state x has not been provided.
`*** ERROR: INCONSISTENT SPECIFICATION OF FAST TRANSITIONS AT STATE n.` — When mixing FAST exponentials with a general fast transition (i.e. using conditional parameters) from a state it is possible to do so in an inconsistent manner. SEE SURE TP.
`*** ERROR: INSTANTANEOUS TRANSITION AT STATE n.` — One of the transitions from state n has been defined with a mean of zero.
`*** ERROR: SUM OF EXITING PROBABILITIES IS NOT 1 AT STATE n.` — The sum of the transition probabilities of the fast transitions from state n does not add up to 1.
`*** ERROR: THE FAST EXPONENTIALS HAVE ZERO PROBABILITY OF OCCURRENCE AT STATE n.` — State n containing mixed fast transition specifications (i.e. some described by FAST exponentials and some by conditional parameters) has been over-specified such that the FAST exponential recoveries have zero probability of occurrence. This occurs when the sum of the transition probabilities of the transitions described by conditional parameters is 1.
`*** ILLEGAL STATE NUMBER` — The state number is negative or greater than the maximum state limit (Default = 10,000, set at SURE compilation time).
`*** STATE x HOLDING TIME ALREADY ENTERED` — The LEE-mode, holding-time information for state x has al ready been entered.
`*** THE *CALC* EXPRESSION MUST BE ON 1 LINE` — The mathematical expression processed by the CALC func tion must fit on one line. Constant sub-expressions can be defined prior to the CALC function and used to simplify the CALC expression.
`*** TRANSITION X -> Y ALREADY ENTERED` — The user is attempting to reenter the same transition again.
`*** VARIABLES INCONSISTENT BETWEEN RUNS` — the ORPROB command cannot process the preceding runs since they did not use the same variable or the same values of the variable.

`*** WARNING: REMAINDER OF INPUT LINE IGNORED` — Any commands that followed the READ command on the same line were ignored.
`*** WARNING: RUN-TIME PROCESSING ERRORS` — Computation overflow occurred during execution.
`*** WARNING: SYNTAX ERRORS PRESENT BEFORE RUN` — Syntax errors were present during the model description process.
`*** WARNING: VARIABLE CHANGED!` — If previous transitions have been defined using a variable and the variable name is changed, inconsistencies can result in the values of the transitions.
`= EXPECTED` — Syntax error; the = operator is needed.
`> EXPECTED` — Syntax error; the closing bracket ¿ is missing.
`) EXPECTED` — A right parenthesis is missing in the expression.
`] EXPECTED` — A right bracket is missing in the expression.

References

[1] Butler, Ricky W.: The SURE Approach to Reliability Analysis. *IEEE Transactions on Reliability*, vol. 41, no. 2, June 1992, pp. 210–218.

[2] Butler, Ricky W.; and White, Allan L.: *SURE Reliability Analysis: Program and Mathematics.* NASA Technical Paper 2764, Mar. 1988.

[3] Johnson, Sally C.: *ASSIST User Manual.* NASA Technical Memorandum 4592, Aug. 1995.