
SCIENTIFIC NOTEBOOK # 846E

Markov Modeling of the Reliability of Complex Engineering Systems

by

Fernando Ferrante

Southwest Research Institute
Center for Nuclear Waste Regulatory
Analyses
San Antonio, Texas

January 18, 2007

Table of Contents

1	Initial Entries.....	5
1.1	Objectives	5
1.2	Computer and Computer Codes	6
2	In-Process Entries	7
2.1	Markov Chain Simulation Basics	7
2.2	Stochastic Transitional Probability Matrix.....	9
2.3	Extension of Discrete Markov Approach to 3-State System	11
2.4	Results from Markov Stochastic Simulation Techniques.....	13
2.5	Development of Uncertainty Analysis in Markov Analytical Estimation	26
2.6	Fault Tree Analysis Comparison	30
2.7	Deterministic Markov Analytical Results	31
2.8	Closure of Scientific Notebook SN846E.....	32
3	References.....	33

List of Figures

1	2-State system probabilities for 10 intervals.....	8
2	Configuration of hypothetical HVAC System	13
3	Uncertainty Analysis in Markov Analytical Approach for the hypothetical HVAC System in Figure 4	29

List of Tables

1	Computers, operating systems, and compilers used.....	6
2	Results of uncertainty analysis performed on Analytical Markov Model.	25

1. INITIAL ENTRIES

Scientific Note Book: # 846E

Issued to: Fernando Ferrante (Initials: FF)

Issue Date: January 18, 2007

Project Title: Markov Modeling of the Reliability of Complex Engineering Systems

Project Staff: Fernando Ferrante (CNWRA), George Adams (CNWRA)

Qualification requirements for this project are: reliability modeling, stochastic processes, system modeling, probabilistic programming capabilities (MATLAB plus Statistic Toolbox, EXCEL).

1.1 Objectives

Redundant components, subsystems and control mechanisms are part of the design of highly reliable engineering systems used in several industrial applications [1]. Such complex engineering systems do not lend themselves well to commonly used reliability modeling techniques (such as fault tree and event tree analysis) because of the inherent dependencies and interactions among redundant components and operating states [2].

These component failures not only may be strongly dependent, but they may even occur when the system is already operating at reduced capacity (e.g., partially shutdown for maintenance). Using such methods may lead to inaccurate estimates of the performance of complex system design, which may be overcome by the implementation of Markov Methods.

The objective of this work is to develop the capability to perform Reliability Modeling using Markov Methods for the development of reliability estimates and its implementation to complex engineering systems. As a first step, the work will be aimed at the power generation industry (e.g., heating and ventilation, electrical supply, control systems) and subsequently extended to other areas if suitable. The analyses is also intended to identify failure modes and modeling of the system states and compare the results to those from other modeling techniques.

1.2 Computers and Computer Codes

MATLAB (Version 7.1.0.246 R14 Service Pack3, License Number 301039) is used to generate random numbers and perform supporting calculations. MATLAB is a general purpose programming environment, developed by MathWorks, suitable for the calculations required in this project. MATLAB's plotting capabilities will also be used to present results. MathCad, developed by MathSoft (Version 2000 Professional) may also be used to check simple calculations, if deemed necessary.

Table 1. Computers, operating systems, and compilers used.

Machine Name	Machine Type	Operating System	Compiler	Location
GRIFFON	PC Desktop	Windows 2000	Intel Pentium	Bldg. 61

[Fernando Ferrante, January 18, 2007]

2. IN-PROCESS ENTRIES

2.1 Markov Chain Simulation Basics

Markov Chains can be advantageous in the reliability analysis of repairable systems due to the capability in modeling repair process usually assumed to be instantaneous or negligible to the overall reliability of the system. Requirements for the applicability of the approach: (i) stationarity, (ii) lack of memory, and (iii) identifiable system states.

The two subroutines below are used to illustrate the generation of a discrete Markov Chain based on the simple example provided by Billinton and Allan [1] in *“Reliability Evaluation of Engineering Systems, Concepts and Techniques”* (page 264-265, Chapter 8). The function “trialstate.m” generated a Markov chain in which 2 possible states are defined (1 and 2) with 0.5 probability of staying in state 1 and 0.75 probability of staying in state 2 (changes of state are complements of these two values). These probabilities are implemented in the function “dice.m” in order to simulate the change (or not) of state. An example output of a generated Markov Chain is shown below:

```
state = [1  2  2  2  2  1  1  2  2  2];
```

trialstate.m

```
function state = trialstate(state1,n)

state(1) = state1;

for i = 2:n
    if state(i-1) == 1
        state(i) = dice(0.5);
    elseif state(i-1) == 2
        state(i) = dice(0.25);
    end
end
```

dice.m

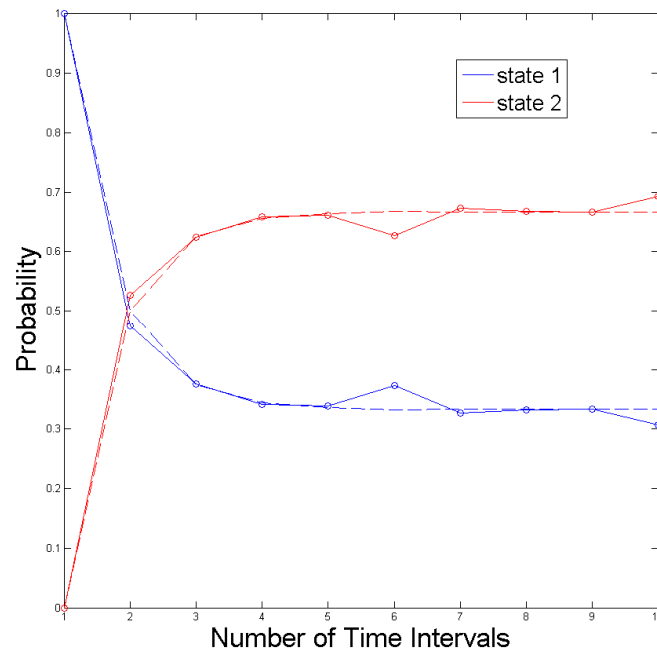
```
function y = dice(lim);

x = rand;
if x < lim
    y = 1;
else
    y = 2;
end
```

In order to compare the results from simulation to the predicted results from reference [1] (page 263 and 270), the routine “Example8p1.m” below is developed with results shown in Figure 1 for

1000 Markov Chains from which the probability of being in State 1 or 2 at different time intervals is obtained. As shown, the results closely match the target (dashed lines) values and they converge as the ensemble becomes large.

Figure 1: 2-State system probabilities for 10 intervals



Example8p1.m:

```
clear all;

N = 1000;
T = 10;
for i = 1:N;
    state(i,:) = trialstate(1,T);
end
for i = 1:T
    P(i) = length(find(state(:,i) == 1))/N;
    PC(i) = length(find(state(:,i) == 2))/N;
end

plot(1:T,P,'-');hold on;plot(1:T,PC,'r-');
legend('state 1','state 2')
plot(1:T,P,'o');plot(1:T,PC,'ro');
ylabel('Probability','FontSize',26);
xlabel('Number of Time Intervals','FontSize',26);
axis square;axis([1 T 0 1])

P1 = [1 1/2 3/8 11/32 43/128 171/515 0.333 0.333 0.333 0.333];
P2 = 1 - P1;
plot(1:T,P1,'b--');plot(1:T,P2,'r--');
```

[Fernando Ferrante, January 23, 2007]

2.2 Stochastic Transitional Probability Matrix

The simulation results can be compared to system reliability evaluation through matrix techniques with a descriptor of the transitional probabilities between the distinct states of the system. The elements P_{ij}^n of the stochastic transitional probability matrix (STPM) can be defined as the probability that the system will be in state j after n intervals given that it started at state i [1]. The further generalization of this approach is to allow for the initial state itself to be random, so that the resulting STPM is pre-multiplied by a vector that contains the probabilities of the potential initial state.

This approach is implemented in the function 'markovc2.m', for a two-state system. Also, this routine includes a calculation for the Limiting State Probability (LSP) which represents the steady-state values of the resulting probabilities of a system being in state 1 or 2 (i.e. when convergence is reached) (see pages 268-270 in [1]). This will be achieved by simulation once a sufficient number of intervals are performed (small for simple systems, potentially larger for complex systems) and can be used as further comparison. Also, the possibility of including an absorbing state (i.e. a state where that cannot be left once entered) is included (see pages 270-272 in [1]). In this case, 'abs' represents the absorbing state (either 1 or 2 for a 2-state system) and the output N is the average number of intervals it will take before the absorbing state 'abs' is entered. The sample calculations below are used to verify this routine:

For the example in page 268 [1], the following output is obtained in MATLAB (note that the input STPM is entered accordingly):

```
[PN,LSP,N] = markovc2(1,[1 0])
```

```
PN = [0.3750  0.6250]
```

```
LSP = [0.3333  0.6667]
```

Which coincides with the results obtained in [1] (see pages 268 and 270). The absorbing state calculation is verified by the result in page 272, with input STPM modified to the one in page 269 in [1], as shown below:

```
[PN,LSP,N] = markovc2(1,[0 1],2)
```

```
PN = [0.2500  0.7500]
```

```
LSP = [0.3333  0.6667]
```

```
N = 2
```

markovc2.m:

```
function [PN,LSP,N] = markovc2(n,P0,abs)

% Input Stochastic Transitional Probability Matrix (2-by-2)
P11 = 3/8;
P12 = 1 - P11;
P21 = 5/16;
P22 = 1 - P21;
P = [P11 P12;
     P21 P22];

% Pre-multiply by initial state probability
PN = P0*(P^n);

% Calculate Limiting State Probability (LSP)
PPP = P' - eye(2);
PPP(2,:) = [1 1];
LSP = inv(PPP)*[0;1];
LSP = LSP';

if nargin == 3
    state = 1:2;
    PP = P(find(state~=abs),find(state~=abs));
    N = inv(eye(1) - PP);
else
    N = 0;
end
```

[Fernando Ferrante, January 25, 2007]

2.3 Extension of Discrete Markov Approach to 3-State System

The previous simple routines can be easily extended to a 3-state system, in anticipation that most complex systems will clearly have more than just 2-states. While a general n-state discrete Markov approach can be implemented, this will be deferred to the continuous Markov (i.e. Markov Processes) where the applications of interest are for this project. The routines 'dice3.m' and 'trialstate3.m' refer to the simulation approach while 'markovc3.m' calculates the LSP (and N, if needed) for an input 3-by-3 STPM.

This approach is implemented in the function 'markovc2.m', for a two-state system. Also, this routine includes a calculation for the Limiting State Probability (LSP) which represents the steady-state values of the resulting probabilities of a system being in state 1 or 2 (i.e. when convergence is reached) (see pages 268-270 in [1]). This will be achieved by simulation once a sufficient number of intervals are performed (small for simple systems, potentially larger for complex systems) and can be used as further comparison. Also, the possibility of including an absorbing state (i.e. a state where that cannot be left once entered) is included (see pages 270-272 in [1]). In this case, 'abs' represents the absorbing state (either 1 or 2 for a 2-state system) and the output N is the average number of intervals it will take before the absorbing state 'abs' is entered. The sample calculations below are used to verify this routine:

For the example in pages 272-274 [1], the following output is obtained in MATLAB (again, note that the input STPM is entered accordingly in the function 'markovc3.m' below):

LSP = [0.3636 0.3636 0.2727]

N = [4 2
0 2]

dice3.m:

```
function y = dice3(lim1,lim2);

%starting with state 1
x = rand;
if x < lim1
    y = 3;
elseif ((x < lim2)&(x > lim1))
    y = 2;
elseif x > lim2
    y = 1;
end
```

trialstate3.m:

```
function state = trialstate3(state1,n)

state(1) = state1;
for i = 2:n
    if state(i-1) == 1
        state(i) = dice3(1/8,0.5);
    elseif state(i-1) == 2
        state(i) = dice3(1/4,0.5);
    end
```

```
elseif state(i-1) == 3
    state(i) = dice3(1/4,0.5);
end
end
```

markovc3.m:

```
function [PN,LSP,N] = markovc3(n,P0,abs)

P12 = 0.25;
P11 = 1 - P12;
P13 = 0;
P21 = 0;
P23 = 0.5;
P22 = 1 - P23;
P32 = 1/3;
P31 = 1/3;
P33 = 1 - P32 - P31;

P = [P11 P12 P13;
     P21 P22 P23;
     P31 P32 P33];

PN = P0*(P^n);

PPP = P' - eye(3);
PPP(3,:) = [1 1 1];
LSP = inv(PPP)*[0;0;1];
LSP = LSP';

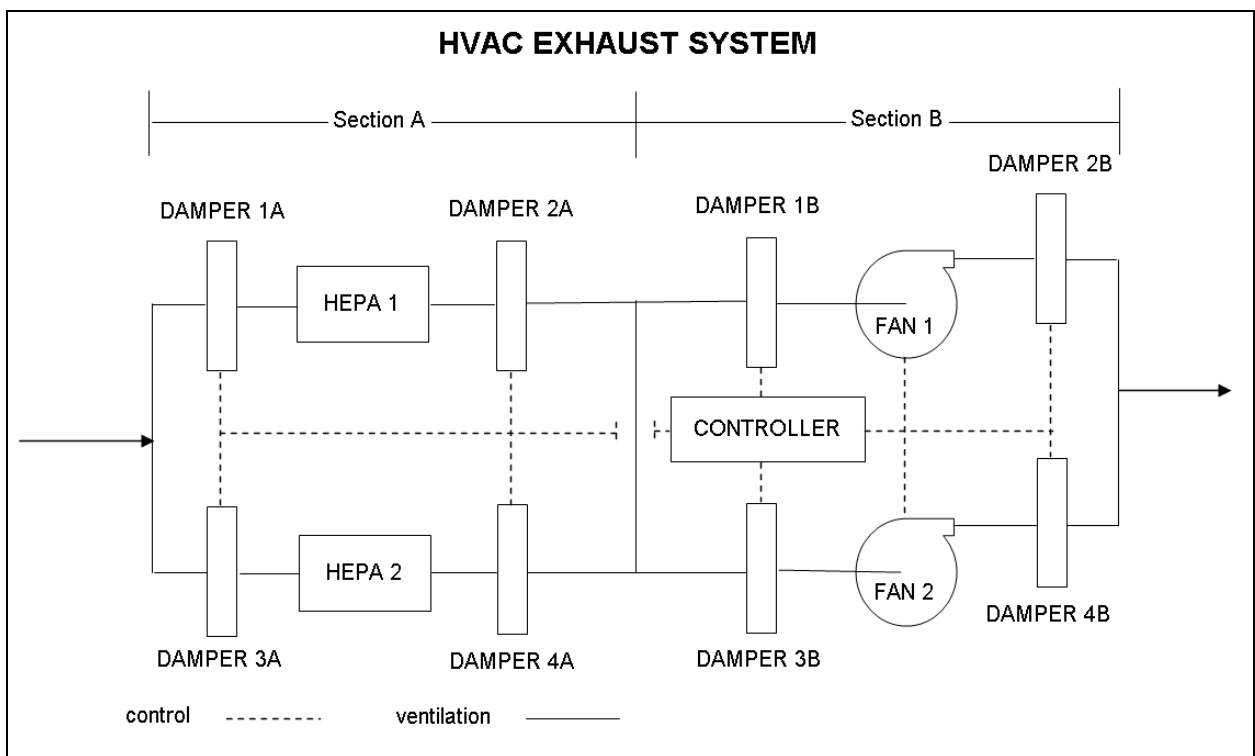
if nargin == 3
    state = 1:3;
    PP = P(find(state~=abs),find(state~=abs));
    N = inv(eye(2) - PP);
else
    N = 0;
end
```

[Fernando Ferrante, January 29, 2007]

2.4 Results from Markov Stochastic Simulation Techniques

The analytical approach described previously can be implemented by performing a real-time stochastic simulation of the operational states of the system. This is equivalent to performing a Monte Carlo simulation where the transition between the assumed discrete states is sampled based on the prescribed failure rates and configuration of the system. The system to be considered is shown in Figure 2, and is an assumed configuration developed by George Adams for an Heating, Ventilation, and Air Conditioning (HVAC) system.

Figure 2: Configuration of hypothetical HVAC System



The failure rates for the components of the HVAC system, the assumed possible system states and maintenance considerations are provided in the Excel spreadsheet 'HVAC.xls' developed and provided by George Adams. The stochastic simulation results are to be obtained using the MATLAB files 'simulation.m', along with subroutines 'mTime.m' and 'fTime.m' (all provided by George Adams).

simulation.m:

```

%*****
% Simulation Model for HVAC failure
% Author: George Adams
% Date: December 8, 2006
%*****

format short e;

% This simulation code is designed to produce a success/failure chain for
% each piece of equipment

% Simulation variables
Total_Simulations = 1000;
Time_Period = 50.0; % (year)

% Failure rates are in failures/year
FD = 2.62E-2; % Failure rate of damper
FF = 2.62E-1; % Failure rate of fan
FC = 2.10E-2; % Failure rate of controller

% Repair/Restore rate is in repairs/year
RR = 2184; % 4 hour mean time to repair/restore
%RR = 1092; % 8 hour mean time to repair/restore
%RR = 728; % 12 hour mean time to repair/restore
%RR = 364; % 24 hour mean time to repair/restore
%RR = 182; % 48 hour mean time to repair/restore

% Maintenance rate is in maintenance period/year
MR = 46.22;

%*****
% Transition Rate Matrices
%*****
% Controller transition matrix
TRC = [-1, FC;
        RR, -1];
StatusC = [2]; % Status: -2 = maintenance, -1 = failed, 1 = standby, 2 = operating

% Section A transition matrix
%
% 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15
TRA = [-1, FD, FD, FD, FD, MR, MR, -1, -1, -1, -1, -1, -1, -1, -1; %ROW 1
        RR, -1, -1, -1, -1, -1, -1, FD, FD, -1, -1, -1, -1, -1, -1; %ROW 2
        RR, -1, -1, -1, -1, -1, -1, -1, -1, FD, FD, -1, -1, -1, -1; %ROW 3
        RR, -1, -1, -1, -1, -1, -1, -1, FD, -1, FD, -1, -1, -1, -1; %ROW 4
        RR, -1, -1, -1, -1, -1, -1, -1, FD, -1, FD, -1, -1, -1, -1; %ROW 5
        RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, FD, FD, -1, -1; %ROW 6
        RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, FD, FD; %ROW 7
        -1, RR, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1; %ROW 8
        -1, RR, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1; %ROW 9
        -1, -1, RR, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1; %ROW 10
        -1, -1, RR, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1; %ROW 11
        -1, -1, -1, RR, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1; %ROW 12
        -1, -1, -1, -1, RR, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1; %ROW 13
        -1, RR, -1, -1, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1; %ROW 14
        -1, -1, RR, -1, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1]; %ROW 15

StatusA = [2, 2; % Status: -2 = maintenance, -1 = failed, 1 = standby, 2 = operating
           1, 1]; % Initially, train 1 is operating and train 2 is in standby

% Section B transition matrix
%
% 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
TRB = [-1, FD, FD, FF, FD, FD, FF, MR, MR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
        %ROW 1
        RR, -1, -1, -1, -1, -1, -1, -1, -1, FD, FD, FF, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
        %ROW 2
        RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, FD, FD, FF, -1, -1, -1, -1, -1, -1, -1, -1, -1;
        %ROW 3
        RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, FD, FD, FF, -1, -1, -1, -1, -1, -1;
        %ROW 4
        RR, -1, -1, -1, -1, -1, -1, -1, -1, FD, -1, -1, FD, -1, -1, FF, -1, -1, -1, -1, -1, -1, -1;
        %ROW 5
        RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, FD, -1, -1, FD, -1, -1, FF, -1, -1, -1, -1, -1, -1;
        %ROW 6
        RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, FD, -1, -1, FD, -1, -1, FF, -1, -1, -1, -1, -1, -1;
        %ROW 7
        RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, FD, FD, FF, -1, -1, -1;
        %ROW 8

```

```

        RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, FD, FD, FF;
%ROW 9
        -1, RR, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 10
        -1, RR, -1, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 11
        -1, RR, -1, -1, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 12
        -1, -1, RR, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 13
        -1, -1, RR, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 14
        -1, -1, RR, -1, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 15
        -1, -1, -1, RR, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 16
        -1, -1, -1, RR, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 17
        -1, -1, -1, RR, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 18
        -1, -1, -1, -1, RR, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 19
        -1, -1, -1, -1, -1, RR, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 20
        -1, -1, -1, -1, -1, -1, RR, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 21
        -1, RR, -1, -1, -1, -1, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 22
        -1, -1, RR, -1, -1, -1, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 23
        -1, -1, -1, RR, -1, -1, -1, -1, -1, RR, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1;
%ROW 24
1]; %ROW 24

StatusB = [2, 2, 2; % Status: -2 = maintenance, -1 = failed, 1 = standby, 2 = operating
           1, 1, 1]; % Initially, train 1 is operating and train 2 is in standby
               % Order is Damper, Damper, Fan in rows of the matrix

% Simulation variables
index_sim = 1;
Simulation = 1;
num_failures_total = 0;
num_fail_controller_total = 0;
num_fail_sectiona_total = 0;
num_fail_sectionb_total = 0;
%xlswrite('simulation.xls', ...
%      {'Simulation', 'No. Failures', 'Controller', 'Section A', 'Section B'}, 'simulation', 'A1');

% Initialize the output file
%xlswrite('states.xls', ...
%      {'Time', 'State C', 'State A', 'State B'}, 'states', 'A1');

num_states = 0;

while index_sim <= Total_Simulations
% Setup the initial state for the simulation
StateC = 1;
LastStateC = 1;
StateA = 1;
LastStateA = 1;
StateB = 1;
LastStateB = 1;
MTime = 0.0;
state = [StateC, StateA, StateB]; %Normal Operation

boperating = true;
boperatingc = true;
boperatinga = true;
boperatingb = true;

StatusC = [2];
StatusA = [2, 2; % Status: -2 = maintenance, -1 = failed, 1 = standby, 2 = operating
           1, 1];
StatusB = [2, 2, 2; % Status: -2 = maintenance, -1 = failed, 1 = standby, 2 = operating
           1, 1, 1];

% Set the seed value in the random number generator
rand('state', sum(100 * clock))

```

```

% The state identifies the row of the transition matrix
RowC = TRC(state(1), :);
RowA = TRA(state(2), :);
RowB = TRB(state(3), :);

% Get sample times from the rows of the transition matrix
[TimeC, IndexC, MTimeC] = fTime(RowC, [], []);
[TimeA, IndexA, MTimeA] = fTime(RowA, [], []);
[TimeB, IndexB, MTimeB] = fTime(RowB, [], []);

remaining_time = Time_Period;
num_states = 0;
num_failures = 0;

num_fail_controller = 0;
num_fail_sectiona = 0;
num_fail_sectionb = 0;

while remaining_time > 0

    if state(1) == 2
        if boperating
            num_failures = num_failures + 1;
            boperating = false;

            num_states = num_states + 1;
            range = strcat('A', int2str(num_states + 1));
            %xlswrite('states.xls', [Time_Period - remaining_time, state, num_failures], ...
                % 'states', range);
        end
    elseif state(2) >= 8
        if boperating
            num_failures = num_failures + 1;
            boperating = false;

            num_states = num_states + 1;
            range = strcat('A', int2str(num_states + 1));
            %xlswrite('states.xls', [Time_Period - remaining_time, state, num_failures], ...
                % 'states', range);
        end
    elseif state(3) >= 10
        if boperating
            num_failures = num_failures + 1;
            boperating = false;

            num_states = num_states + 1;
            range = strcat('A', int2str(num_states + 1));
            %xlswrite('states.xls', [Time_Period - remaining_time, state, num_failures], ...
                % 'states', range);
        end
    else
        boperating = true;
    end

    if state(1) == 2
        if boperatingc
            num_fail_controller = num_fail_controller + 1;
            boperatingc = false;
        end
    else
        boperatingc = true;
    end

    if state(2) >= 8
        if boperatinga
            num_fail_sectiona = num_fail_sectiona + 1;
            boperatinga = false;
        end
    else
        boperatinga = true;
    end

    if state(3) >= 10
        if boperatingb
            num_fail_sectionb = num_fail_sectionb + 1;
            boperatingb = false;
        end
    else

```

```

        boperatingb = true;
    end

% Determine the next state for the system
[MTime, IndexMin] = min([MTimeC, MTimeA, MTimeB]);

% Controller portion of the state
if IndexMin == 1
    StateC = IndexC;
    ResidualsC = [];
    StationaryC = [];

    if StateC == 1
        % In state 1, the controller is operating
        StatusC(1) = 2;
    elseif StateC == 2
        % In state 2, the controller has failed
        StatusC(1) = -1;
    end
    state = [StateC, StateA, StateB];
    RowC = TRC(state(1), :);
    TimeA = TimeA - MTime;
    MTimeA = mTime(TimeA);
    TimeB = TimeB - MTime;
    MTimeB = mTime(TimeB);
    [TimeC, IndexC, MTimeC] = fTime(RowC, ResidualsC, StationaryC);
elseif IndexMin == 2
    LastStateA = StateA;
    StateA = IndexA;
    ResidualsA = [];
    StationaryA = [];

    switch StateA
        case 1
            if ((StatusA(1,1) == 2) && (StatusA(1,2) == 2))
                switch LastStateA
                    case 4
                        ResidualsA = [(TimeA(8)-MTime), (TimeA(10)-MTime)];
                    case 5
                        ResidualsA = [(TimeA(9)-MTime), (TimeA(11)-MTime)];
                    case 7
                        ResidualsA = [(TimeA(14)-MTime), (TimeA(15)-MTime)];
                    otherwise
                        disp('Error: StateA, Cases 4, 5, 7');
                end
                StationaryA = [4, 5];
                StatusA = [2, 2;
                           1, 1];
            else
                switch LastStateA
                    case 2
                        ResidualsA = [(TimeA(8)-MTime), (TimeA(9)-MTime);
                                       4, 5];
                    case 3
                        ResidualsA = [(TimeA(10)-MTime), (TimeA(11)-MTime);
                                       4, 5];
                    case 6
                        ResidualsA = [(TimeA(12)-MTime), (TimeA(13)-MTime);
                                       4, 5];
                    otherwise
                        disp('Error: StateA, Cases 2, 3, 6');
                end
                StationaryA = [2, 3];
                StatusA = [1, 1;
                           2, 2];
            end
        case 2
            StatusA = [-1, 1;
                       2, 2];
            switch LastStateA
                case 8
                    ResidualsA = [(TimeA(4)-MTime);1];
                case 9
                    ResidualsA = [(TimeA(5)-MTime);1];
                case 14
                    ResidualsA = [(TimeA(7)-MTime);1];
            end
        case 3
            StatusA = [1, -1;
                       2, 2];

```

```

switch LastStateA
case 10
    ResidualsA = [(TimeA(4)-MTime);1];
case 11
    ResidualsA = [(TimeA(5)-MTime);1];
case 15
    ResidualsA = [(TimeA(7)-MTime);1];
end
case 4
    StatusA = [2, 2;
               -1, 1];
    switch LastStateA
    case 8
        ResidualsA = [(TimeA(2)-MTime);1];
    case 10
        ResidualsA = [(TimeA(3)-MTime);1];
    case 12
        ResidualsA = [(TimeA(6)-MTime);1];
    end
case 5
    StatusA = [2, 2;
               1, -1];
    switch LastStateA
    case 9
        ResidualsA = [(TimeA(2)-MTime);1];
    case 11
        ResidualsA = [(TimeA(3)-MTime);1];
    case 13
        ResidualsA = [(TimeA(6)-MTime);1];
    end
case 6
    if LastStateA == 1
        if ((StatusA(2,1) == 2) && (StatusA(2,2) == 2))
            ResidualsA = [(TimeA(4)-MTime), (TimeA(5)-MTime);
                           12,                13];
        end
    else
        switch LastStateA
        case 12
            ResidualsA = [(TimeA(4)-MTime);1];
        case 13
            ResidualsA = [(TimeA(5)-MTime);1];
        end
    end
    StatusA = [-2, -2;
               2, 2];
case 7
    if LastStateA == 1
        if ((StatusA(1,1) == 2) && (StatusA(1,2) == 2))
            ResidualsA = [(TimeA(2)-MTime), (TimeA(3)-MTime);
                           14,                15];
        end
    else
        switch LastStateA
        case 14
            ResidualsA = [(TimeA(2)-MTime);1];
        case 15
            ResidualsA = [(TimeA(3)-MTime);1];
        end
    end
    StatusA = [ 2, 2;
               -2, -2];
case 8
    StatusA = [-1, 1;
               -1, 1];
    switch LastStateA
    case 2
        ResidualsA = [(TimeA(1)-MTime);4];
    case 4
        ResidualsA = [(TimeA(1)-MTime);2];
    end
case 9
    StatusA = [-1, 1;
               1, -1];
    switch LastStateA
    case 2
        ResidualsA = [(TimeA(1)-MTime);5];
    case 5
        ResidualsA = [(TimeA(1)-MTime);2];
    end
end

```

```

    case 10
        StatusA = [ 1, -1;
                   -1, 1];

        switch LastStateA
            case 3
                ResidualsA = [(TimeA(1)-MTime);4];
            case 4
                ResidualsA = [(TimeA(1)-MTime);3];
            end
    case 11
        StatusA = [1, -1;
                   1, -1];
        switch LastStateA
            case 3
                ResidualsA = [(TimeA(1)-MTime);5];
            case 5
                ResidualsA = [(TimeA(1)-MTime);3];
            end

    case 12
        StatusA = [-2, -2;
                   -1, 1];
        ResidualsA = [(TimeA(1)-MTime); 4];
    case 13
        StatusA = [-2, -2;
                   1, -1];
        ResidualsA = [(TimeA(1)-MTime); 5];
    case 14
        StatusA = [-1, 1;
                   -2, -2];
        ResidualsA = [(TimeA(1)-MTime); 2];
    case 15
        StatusA = [ 1, -1;
                   -2, -2];
        ResidualsA = [(TimeA(1)-MTime); 3];
    otherwise
        disp('Error: IndexMin = 2');
end
state = [StateC, StateA, StateB];
RowA = TRA(state(2), :);
TimeC = TimeC - MTime;
MTimeC = mTime(TimeC);
TimeB = TimeB - MTime;
MTimeB = mTime(TimeB);
[TimeA, IndexA, MTimeA] = fTime(RowA, ResidualsA, StationaryA);
elseif IndexMin == 3
    LastStateB = StateB;
    StateB = IndexB;
    ResidualsB = [];
    StationaryB = [];
    switch StateB
        case 1
            if ((StatusB(1,1) == 2) && (StatusB(1,2) == 2) && (StatusB(1,3) == 2))
                switch LastStateB
                    case 5
                        ResidualsB = [(TimeB(10)-MTime), (TimeB(13)-MTime), (TimeB(16)-MTime);
                                     2, 3, 4];
                    case 6
                        ResidualsB = [(TimeB(11)-MTime), (TimeB(14)-MTime), (TimeB(17)-MTime);
                                     2, 3, 4];
                    case 7
                        ResidualsB = [(TimeB(12)-MTime), (TimeB(15)-MTime), (TimeB(18)-MTime);
                                     2, 3, 4];
                    case 9
                        ResidualsB = [(TimeB(22)-MTime), (TimeB(23)-MTime), (TimeB(24)-MTime);
                                     2, 3, 4];
                    otherwise
                        disp('Error: StateB, Cases 5, 6, 7, and 9');
                end
            end
            StationaryB = [5, 6, 7];

            StatusB = [2, 2, 2;
                      1, 1, 1];
        else
            switch LastStateB
                case 2
                    ResidualsB = [(TimeB(10)-MTime), (TimeB(11)-MTime), (TimeB(12)-MTime);
                                 5, 6, 7];
            end
        end
    end
end

```

```

        case 3
            ResidualsB = [(TimeB(13)-MTime), (TimeB(14)-MTime), (TimeB(15)-MTime);
                          5,                  6,                  7];
        case 4
            ResidualsB = [(TimeB(16)-MTime), (TimeB(17)-MTime), (TimeB(18)-MTime);
                          5,                  6,                  7];
        case 8
            ResidualsB = [(TimeB(19)-MTime), (TimeB(20)-MTime), (TimeB(21)-MTime);
                          5,                  6,                  7];
        otherwise
            disp('Error: StateB, Cases 2, 3, 4, and 8');
        end

        StationaryB = [2, 3, 4];

        StatusB = [1, 1, 1;
                   2, 2, 2];

    end

case 2
    StatusB = [-1, 1, 1;
               2, 2, 2];

    switch LastStateB
        case 10
            ResidualsB = [(TimeB(5)-MTime);1];
        case 11
            ResidualsB = [(TimeB(6)-MTime);1];
        case 12
            ResidualsB = [(TimeB(7)-MTime);1];
        case 22
            ResidualsB = [(TimeB(9)-MTime);1];
    end

case 3
    StatusB = [1, -1, 1;
               2, 2, 2];

    switch LastStateB
        case 13
            ResidualsB = [(TimeB(5)-MTime);1];
        case 14
            ResidualsB = [(TimeB(6)-MTime);1];
        case 15
            ResidualsB = [(TimeB(7)-MTime);1];
        case 23
            ResidualsB = [(TimeB(9)-MTime);1];
    end

case 4
    StatusB = [1, 1, -1;
               2, 2, 2];

    switch LastStateB
        case 16
            ResidualsB = [(TimeB(5)-MTime);1];
        case 17
            ResidualsB = [(TimeB(6)-MTime);1];
        case 18
            ResidualsB = [(TimeB(7)-MTime);1];
        case 24
            ResidualsB = [(TimeB(9)-MTime);1];
    end

case 5
    StatusB = [ 2, 2, 2;
               -1, 1, 1];

    switch LastStateB
        case 10
            ResidualsB = [(TimeB(2)-MTime);1];
        case 13
            ResidualsB = [(TimeB(3)-MTime);1];
        case 16
            ResidualsB = [(TimeB(4)-MTime);1];
        case 19
            ResidualsB = [(TimeB(8)-MTime);1];
    end

case 6

```

```

StatusB = [2, 2, 2;
           1, -1, 1];

switch LastStateB
case 11
    ResidualsB = [(TimeB(2)-MTime);1];
case 14
    ResidualsB = [(TimeB(3)-MTime);1];
case 17
    ResidualsB = [(TimeB(4)-MTime);1];
case 20
    ResidualsB = [(TimeB(8)-MTime);1];
end

case 7
StatusB = [2, 2, 2;
           1, 1, -1];

switch LastStateB
case 12
    ResidualsB = [(TimeB(2)-MTime);1];
case 15
    ResidualsB = [(TimeB(3)-MTime);1];
case 18
    ResidualsB = [(TimeB(4)-MTime);1];
case 21
    ResidualsB = [(TimeB(8)-MTime);1];
end

case 8

if LastStateB == 1
    if ((StatusB(2,1) == 2) && (StatusB(2,2) == 2) && (StatusB(2,3) == 2))
        ResidualsB = [(TimeB(5)-MTime), (TimeB(6)-MTime), (TimeB(7)-MTime);
                       19,                20,                21];
    end
else
    switch LastStateB
    case 19
        ResidualsB = [(TimeB(5)-MTime);1];
    case 20
        ResidualsB = [(TimeB(6)-MTime);1];
    case 21
        ResidualsB = [(TimeB(7)-MTime);1];
    end
end

StatusB = [-2, -2, -2;
           2, 2, 2];

case 9

if LastStateB == 1
    if ((StatusB(1,1) == 2) && (StatusB(1,2) == 2) && (StatusB(1,3) == 2))
        ResidualsB = [(TimeB(2)-MTime), (TimeB(3)-MTime), (TimeB(4)-MTime);
                       22,                23,                24];
    end
else
    switch LastStateB
    case 22
        ResidualsB = [(TimeB(2)-MTime);1];
    case 23
        ResidualsB = [(TimeB(3)-MTime);1];
    case 24
        ResidualsB = [(TimeB(4)-MTime);1];
    end
end

StatusB = [ 2, 2, 2;
           -2, -2, -2];

case 10

StatusB = [-1, 1, 1;
           -1, 1, 1];

switch LastStateB

```

```
        case 2
            ResidualsB = [(TimeB(1)-MTime);5];
        case 5
            ResidualsB = [(TimeB(1)-MTime);2];
        end

    case 11
        StatusB = [-1, 1, 1;
                   1, -1, 1];

        switch LastStateB
            case 2
                ResidualsB = [(TimeB(1)-MTime);6];
            case 6
                ResidualsB = [(TimeB(1)-MTime);2];
        end

    case 12
        StatusB = [-1, 1, 1;
                   1, 1, -1];

        switch LastStateB
            case 2
                ResidualsB = [(TimeB(1)-MTime);7];
            case 7
                ResidualsB = [(TimeB(1)-MTime);2];
        end

    case 13
        StatusB = [ 1, -1, 1;
                   -1, 1, 1];

        switch LastStateB
            case 3
                ResidualsB = [(TimeB(1)-MTime);5];
            case 5
                ResidualsB = [(TimeB(1)-MTime);3];
        end

    case 14
        StatusB = [1, -1, 1;
                   1, -1, 1];

        switch LastStateB
            case 3
                ResidualsB = [(TimeB(1)-MTime);6];
            case 6
                ResidualsB = [(TimeB(1)-MTime);3];
        end

    case 15
        StatusB = [1, -1, 1;
                   1, 1, -1];

        switch LastStateB
            case 3
                ResidualsB = [(TimeB(1)-MTime);7];
            case 7
                ResidualsB = [(TimeB(1)-MTime);3];
        end

    case 16
        StatusB = [ 1, 1, -1;
                   -1, 1, 1];

        switch LastStateB
            case 4
                ResidualsB = [(TimeB(1)-MTime);5];
            case 5
                ResidualsB = [(TimeB(1)-MTime);4];
        end

    case 17
        StatusB = [1, 1, -1;
                   1, -1, 1];

        switch LastStateB
            case 4
                ResidualsB = [(TimeB(1)-MTime);6];
            case 6
                ResidualsB = [(TimeB(1)-MTime);4];
```

```

        end

    case 18
        StatusB = [1, 1, -1;
                   1, 1, -1];

        switch LastStateB
            case 4
                ResidualsB = [(TimeB(1)-MTime);7];
            case 7
                ResidualsB = [(TimeB(1)-MTime);4];
        end

    case 19
        StatusB = [-2, -2, -2;
                   -1, 1, 1];

        ResidualsB = [(TimeB(1)-MTime);5];

    case 20
        StatusB = [-2, -2, -2;
                   1, -1, 1];

        ResidualsB = [(TimeB(1)-MTime);6];

    case 21
        StatusB = [-2, -2, -2;
                   1, 1, -1];

        ResidualsB = [(TimeB(1)-MTime);7];

    case 22
        StatusB = [-1, 1, 1;
                   -2, -2, -2];

        ResidualsB = [(TimeB(1)-MTime);2];

    case 23
        StatusB = [ 1, -1, 1;
                   -2, -2, -2];

        ResidualsB = [(TimeB(1)-MTime);3];

    case 24
        StatusB = [ 1, 1, -1;
                   -2, -2, -2];

        ResidualsB = [(TimeB(1)-MTime);4];

    otherwise
        disp('Error: IndexMin = 3');
end

state = [StateC, StateA, StateB];
RowB = TRB(state(3), :);
TimeC = TimeC - MTime;
MTimeC = mTime(TimeC);
TimeA = TimeA - MTime;
MTimeA = mTime(TimeA);
[TimeB, IndexB, MTimeB] = fTime(RowB, ResidualsB, StationaryB);
else
    error_msg = 'Error'
end

remaining_time = remaining_time - MTime;

% Display results
%state
%remaining_time
%num_failures

end

% Display simulation results
index_sim
num_failures

num_failures_total = num_failures_total + num_failures;
num_fail_controller_total = num_fail_controller_total + num_fail_controller;
num_fail_sectiona_total = num_fail_sectiona_total + num_fail_sectiona;
num_fail_sectionb_total = num_fail_sectionb_total + num_fail_sectionb;

```

```

range = strcat('A', int2str(index_sim + 1));
%xlswrite('simulation.xls', ...
%   [index_sim, num_failures, num_fail_controller, num_fail_sectiona, num_fail_sectionb], ...
%   'simulation', range)

index_sim = index_sim + 1;

end

num_failures_total
num_fail_controller
num_fail_sectiona
num_fail_sectionb
average_failure_rate = num_failures_total/(Time_Period * Total_Simulations)

```

fTime.m:

```

function[time, index_min, time_min] = fTime(rate, residual, stationary)

% Initialize the return values
time = rate;
index_min = -1;
time_min = -1.0;

% Determine the number of cells to process
ncells = size(time, 2);
numstat = size(stationary, 2);
numresid = size(residual, 2);

% For each positive value generate a time
for index = 1 : ncells
    if time(index) > 0
        random_value = rand();
        time(index) = -1 * log(random_value) / time(index);
    end
end

% Modify the return matrix for residuals and stationaries
for index = 1 : numresid
    time(residual(2,index)) = residual(1,index);
end

for index = 1 : numstat
    time(stationary(index)) = -1;
end

% Find the minimum time
for index = 1 : ncells
    if time(index) > 0
        if index_min == -1
            index_min = index;
            time_min = time(index);
        elseif time(index) < time_min
            index_min = index;
            time_min = time(index);
        end
    end
end

if time_min <= 0
    error('fTime: time_min is less than or equal to 0');
end

return;
end

```

mTime.m:

```

function [time_min] = mTime(time)

% Initialize the return values
time_min = -1.0;

index_min = -1;

% Determine the number of cells to process
ncells = size(time, 2);

% For each positive value generate a time
for index = 1 : ncells
    if time(index) > 0
        if index_min == -1
            index_min = index;
            time_min = time(index);
        elseif time(index) < time_min
            index_min = index;
            time_min = time(index);
        end
    end
end

if (time_min) <= 0
    error('mTime: time_min is less than or equal to 0');
end
return;
end

```

The results from this file are shown below for several input repair/restore (R/R) rates (in units of repairs/year) corresponding to Mean Time To Repair/Restore values (MTTR/R), and were run in MATLAB (GRIFFON machine). For each input, 1000 simulations were performed for an assumed operating time period of 50 years.

Table 2. Results of uncertainty analysis performed on Analytical Markov Model

MTTR/R (hours)	R/R rates (repairs/year)	Average Failure Rate (failures/year)
4	2184	3.720e-002
8	1092	4.8520e-002
12	728	6.3680e-002
24	364	9.4100e-002
48	182	1.4082e-001

[Fernando Ferrante, February 6, 2007]

2.5 Development of Uncertainty Analysis in Markov Analytical Estimation

A routine is developed in MATLAB to evaluate the effect of uncertainty in the failure rates of the components used in the hypothetical HVAC system described in the previous entry. The input for this analysis is the failure rates used in the analysis provided by George Adams, along with the uncertainty description values available in the same literature they were derived from [3]. For the fan and damper, the referenced document [3] provides the probabilistic distribution (lognormal) and mean and error factors EF associated with them (10 and 3, respectively). For the controller, a lognormal distribution with mean corresponding to the published literature previously referenced are assumed, along with an error factor of 3 [4].

The following routines were built based on the analytical approach provided by George Adams (see Excel spreadsheet 'HVAC.xls'), where random sampling from lognormal distributions for the characteristics described above are used to estimate the range of uncertainty around the mean Markov simulation results.

markovsysrand.m:

```
clear all

format short e

% number of states
no = 9;
% vector of state numbers
state = 1:no;
% zero matrix for population
A = zeros(length(state));

% TIME PARAMETERS
nw = 52;% number of weeks per year
nh = nw*7*24;% number of hours per year
dt = 1e-7;% yearly time increment

% MAINTENANCE PARAMETERS
%MTTRh = [4 8 12 24 48];% mean time to repair or restore per hour
MTTRh = [4:1:48];% mean time to repair or restore per hour
RRy = (1./MTTRh)*nh;% repair or restore rate per year
int = [4 4 8 8 24 24];% interval vector (weeks)
mdur = [4 8 12 24 24 48];% maintenance duration (hours)
nint = nw./int;% number of maintenance intervals
tmdur = mdur.*nint;% total maintenance duration (hours)
inser = nh - sum(tmdur);% in-service duration
mrath = sum(nint)/inser;% maintenance rate per hour
mraty = nh*mrath;% maintenance rate per year

% DAMPER FAILURE RATES
% From: Roy, B.N., Savannah River Site Generic Data Base Development (U),
% WSRC-TR-93-262, Rev. 1, May 1998
Dfrh = 3e-6;% failure to run per hour
dfd = 1e-2;% failure on demand

% FAN FAILURE RATES
% Paula, Henrique Martini, Technical note, Failure rates for programmable
% logic controllers, Reliability Engineering and System Safety, Vol. 39
% (1993) 325-328
Ffrh = 3e-5;% failure to run per hour
ffd = 5e-3;% failure on demand

% PLC FAILURE RATES
Cfry = 0.021;% failure to run per year
```

```

N = 10000;% number of random inputs
EF = 5;% error factor

% RANDOM FAILURE RATES
% DAMPER
EF = 3;
%dfrrh = lognrnd(log(Dfrrh),log(EF)/norminv(0.95),1,N);%median as input
dfrh = lognrnd(log(Dfrrh*exp(-
0.5*(log(EF)/norminv(0.95))^2)),log(EF)/norminv(0.95),1,N);%mean as input
dfry = dfrh*nh;% failure to run per year
% FAN
EF = 10;
%ffrrh = lognrnd(log(Ffrrh),log(EF)/norminv(0.95),1,N);%median as input
ffrh = lognrnd(log(Ffrrh*exp(-
0.5*(log(EF)/norminv(0.95))^2)),log(EF)/norminv(0.95),1,N);%mean as input
ffry = ffrh*nh;% failure to run per year
% CONTROLLER
EF = 3;
%cfry = lognrnd(log(Cfry),log(EF)/norminv(0.95),1,N);%median as input
cfry = lognrnd(log(Cfry*exp(-
0.5*(log(EF)/norminv(0.95))^2)),log(EF)/norminv(0.95),1,N);%mean as input

% CALCULATE EFFECTIVE SYSTEM FAILURE RATE
h = waitbar(0,'Please wait...');
for i = 1:length(MTTRh)
%       h = waitbar(0,'Please wait...');
    t = sprintf('Iteration = %d',MTTRh(i));
    for j = 1:N
        he(i,j) = markovsyseffrand(dt,mraty,RRy(i),dfry(j),ffry(j),cfry(j));
%       waitbar(j/N,h,t)
    end
    waitbar(i/length(MTTRh),h,t)
end
%   close(h)
end
close(h)

figure;

% RESULTS FROM GEORGE ADAMS
MTTR = [4 8 12 24 48];% mean time to repair or restore per hour
FT = [3.6576E-02
5.2153E-02
6.7729E-02
1.1446E-01
2.0792E-01];
plot(MTTR,FT,'ks','MarkerFaceColor','k','MarkerSize',10);hold on
%plot(MTTR,FT,'k-','LineWidth',1)
MKmean = [3.7200E-02
4.8520E-02
6.3680E-02
9.4100E-02
1.41E-01];
plot(MTTR,MKmean,'k^','MarkerFaceColor','k','MarkerSize',10)
%plot(MTTR,MKmean,'k-','LineWidth',1)
MK = [3.5900E-02
4.9500E-02
6.2000E-02
9.4800E-02
1.4600E-01];
plot(MTTR,MK,'ko','MarkerFaceColor','k','MarkerSize',10)

%plot(MTTRh,median(he,2),'-','LineWidth',2);hold on
plot(MTTRh,mean(he,2),'k-','LineWidth',2);hold on
Phea = prctile(he',25,1);
plot(MTTRh,Phea,'k--','LineWidth',2)
Pheb = prctile(he',75,1);
plot(MTTRh,Pheb,'k-.','LineWidth',2)

%legend('Median','Mean','15%','85%');

axis([0 50 0 0.16])

```

```

legend('FAULT TREE','SIMULATION','MARKOV','MEAN','15%','85%');
xlabel('Mean Time to Repair/Restore (hr)','FontSize',26)
ylabel('Estimated Failure Rate Per Year','FontSize',26)

```

markovsyseffrand.m:

```

function he = markovsyseffrand(dt,mraty,RRy,dfry,ffry,cfry);

r1 = RRy*dt;
r2 = 2*RRy*dt;
r3 = (dfry + dfry + 2*mraty)*dt;
r4 = (dfry + dfry + ffry + 2*mraty)*dt;
r5 = cfry*dt;
r6 = (dfry + dfry + ffry + mraty)*dt;
r7 = (dfry + dfry)*dt;
r8 = (dfry + dfry + mraty)*dt;
r9 = (dfry + dfry + ffry)*dt;
r10 = (dfry + dfry + dfry + dfry + ffry + ffry)*dt;

% Input non-zero rate values in stochastic probability matrix
% row 1
A(1,2) = r3; A(1,3) = r4; A(1,5) = r5; A(1,1) = 1-sum(A(1,:));
% row 2
A(2,1) = r1; A(2,4) = r6; A(2,6) = r5; A(2,7) = r7; A(2,2) = 1-sum(A(2,:));
% row 3
A(3,1) = r1; A(3,4) = r8; A(3,6) = r5; A(3,7) = r9; A(3,3) = 1-sum(A(3,:));
% row 4
A(4,2) = r1; A(4,3) = r1; A(4,8) = r5; A(4,9) = r10; A(4,4) = 1-sum(A(4,:));
% row 5
A(5,1) = r1; A(5,5) = 1-sum(A(5,:));
% row 6
A(6,2) = r1; A(6,3) = r1; A(6,5) = r1; A(6,6) = 1 - sum(A(6,:));
% row 7
A(7,2) = r2; A(7,3) = r2; A(7,7) = 1-sum(A(7,:));
% row 8
A(8,4) = r1; A(8,8) = 1-sum(A(8,:));
% row 9
A(9,4) = r2; A(9,9) = 1-sum(A(9,:));

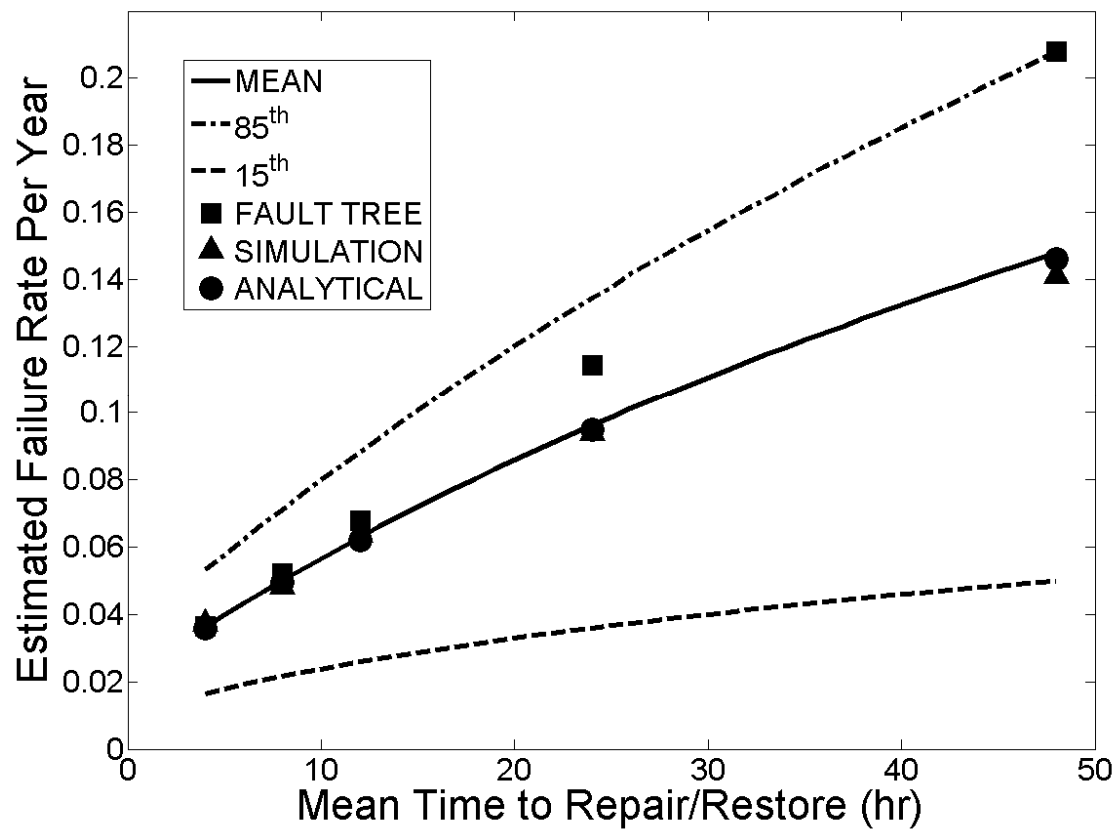
% Obtain matrix with average time spent in state j given start in i
nabs = 1:4;% Define non-absorbing (success) states
A_1 = A(nabs,nabs);
M = inv(eye(size(A_1)) - A_1);

% EFFECTIVE PROPERTIES
MTTFe = (sum(M(1,:))*dt);% effective mean time to failure of the system
he = 1/MTTFe;% effective failure rate of the system per year

```

The results of this routines can be seen in Figure 3, which plots the 15th, 85th percentiles along with the mean value resulting from 10,000 runs. Values from the analytical and simulation Markov analysis are included, along with the results for a fault tree analysis derived from Figure 4 and provided by George Adams.

Figure 3: Uncertainty Analysis in Markov Analytical Approach for the hypothetical HVAC System in Figure 2



[Fernando Ferrante, February 9, 2007]

2.6 Fault Tree Analysis Comparison

A calculation check was done by George Adams in the Excel file 'Fault Tree.xls' and the results are also included in the summary tab of 'HVAC.xls'. These results were compared to the SAPHIRE output, of which the SAPHIRE files used by George Adams are included in CD format.

[Fernando Ferrante, February 20, 2007]

2.7 Deterministic Markov Analytical Results

The deterministic calculation to support the uncertainty analysis was carried in advance and documented in the MATLAB file 'markovsys.m'. This results from this file match the analytical results provided in 'HVAC.xls', and were used to complete the technical review of the ASME POWER07 paper, performed by Amit Ghosh on the date of this entry. By running the file below with a Mean Time To Repair or Restore (MTTRh) of 12 hours, the corresponding effective mean time to failure of the system (MTTFe) is calculated to be 16.12 years, and the effective failure rate of the system per year is 6.2039e-002 failures per year.

markovsys.m:

```
clear all

MTTRh = 12;

format short e

% number of states
no = 9;
% vector of state numbers
state = 1:no;
% zero matrix for population
A = zeros(length(state));

% TIME PARAMETERS
nw = 52;% number of weeks per year
nh = nw*7*24;% number of hours per year
dt = 1e-7;% yearly time increment

% MAINTENANCE PARAMETERS
RRy = (1/MTTRh)*nh;% repair or restore rate per year
int = [4 4 8 8 24 24];% interval vector (weeks)
mdur = [4 8 12 24 24 48];% maintenance duration (hours)
nint = nw./int;% number of maintenance intervals
tmdur = mdur.*nint;% total maintenance duration (hours)
inser = nh - sum(tmdur);% in-service duration
mrath = sum(nint)/inser;% maintenance rate per hour
mraty = nh*mrath;% maintenance rate per year

% DAMPER FAILURE RATES
% From: Roy, B.N., Savannah River Site Generic Data Base Development (U),
% WSRC-TR-93-262, Rev. 1, May 1998
dfrh = 3e-6;% failure to run per hour
dfry = dfrh*nh;% failure to run per year
dfd = 1e-2;% failure on demand

% FAN FAILURE RATES
% Paula, Henrique Martini, Technical note, Failure rates for programmable
% logic controllers, Reliability Engineering and System Safety, Vol. 39
% (1993) 325-328
ffrh = 3e-5;% failure to run per hour
ffry = ffrh*nh;% failure to run per year
ffd = 5e-3;% failure on demand

% PLC FAILURE RATES
cfry = 0.021;% failure to run per year

% STOCHASTIC PROBABILITY MATRIX INPUT VALUES (all of these are per year)
r1 = RRY*dt;
r2 = 2*RRY*dt;
r3 = (dfry + dfry + 2*mraty)*dt;
r4 = (dfry + dfry + ffry + 2*mraty)*dt;
```

```

r5 = cfry*dt;
r6 = (dfry + dfry + ffry + mraty)*dt;
r7 = (dfry + dfry)*dt;
r8 = (dfry + dfry + mraty)*dt;
r9 = (dfry + dfry + ffry)*dt;
r10 = (dfry + dfry + dfry + dfry + ffry + ffry)*dt;

% Input non-zero rate values in stochastic probability matrix
% row 1
A(1,2) = r3; A(1,3) = r4; A(1,5) = r5; A(1,1) = 1-sum(A(1,:));
% row 2
A(2,1) = r1; A(2,4) = r6; A(2,6) = r5; A(2,7) = r7; A(2,2) = 1-sum(A(2,:));
% row 3
A(3,1) = r1; A(3,4) = r8; A(3,6) = r5; A(3,7) = r9; A(3,3) = 1-sum(A(3,:));
% row 4
A(4,2) = r1; A(4,3) = r1; A(4,8) = r5; A(4,9) = r10; A(4,4) = 1-sum(A(4,:));
% row 5
A(5,1) = r1; A(5,5) = 1-sum(A(5,:));
% row 6
A(6,2) = r1; A(6,3) = r1; A(6,5) = r1; A(6,6) = 1 - sum(A(6,:));
% row 7
A(7,2) = r2; A(7,3) = r2; A(7,7) = 1-sum(A(7,:));
% row 8
A(8,4) = r1; A(8,8) = 1-sum(A(8,:));
% row 9
A(9,4) = r2; A(9,9) = 1-sum(A(9,:));

% Obtain limiting state probabilities
% AA = A' - eye(size(A));
% AA(length(A),:) = [ones(1,length(A))];
% AAA = zeros(length(A),1);
% AAA(length(AAA)) = 1;
% A_Final = inv(AA)*AAA;
% A_Final = A_Final';

% Obtain matrix with average time spent in state j given start in i
nabs = 1:4;% Define non-absorbing (success) states
A_1 = A(nabs,nabs);
M = inv(eye(size(A_1)) - A_1);

% EFFECTIVE PROPERTIES
MTTFe = (sum(M(1,:))*dt);% effective mean time to failure of the system
he = 1/MTTFe;% effective failure rate of the system per year

```

[Fernando Ferrante, February 23, 2007]

2.8 Closure of Scientific Notebook SN846E

This scientific notebook was provided to the cognizant manager Asad Chowdhury for closure on June 10, 2008; as required by QAP-01. MATLAB files are provided on a compact disk labeled "Attachment to SN846E".

[Fernando Ferrante, June 10, 2008]

3. References

1. Billinton, R. and R. Allan. *Reliability Evaluation of Engineering Systems, Concepts and Techniques, 2nd edition*. New York City, New York: Plenum Press. 1992.
2. Bouissou, M. and J. Bon. "A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes." *Reliability Engineering & System Safety*. Vol. 82. pp. 149–163. 2003.
3. Roy, B.N., Savannah River Site Generic Data Base Development (U), WSRC-TR-93-262, Rev. 1, May 1998
4. Paula, Henrique Martini, Technical note, Failure rates for programmable logic controllers, *Reliability Engineering and System Safety*, Vol. 39 (1993) 325-328

ADDITIONAL INFORMATION FOR SCIENTIFIC NOTEBOOK NO. 846E

Document Date:	01/18/2008	
Availability:	Southwest Research Institute® Center for Nuclear Waste Regulatory Analyses 6220 Culebra Road San Antonio, Texas 78228	
Contact:	Southwest Research Institute® Center for Nuclear Waste Regulatory Analyses 6220 Culebra Road San Antonio, TX 78228-5166 Attn.: Director of Administration 210.522.5054	
Data Sensitivity:	☒ "Non-Sensitive" ☐ Sensitive ☐ "Non-Sensitive - Copyright" ☐ Sensitive - Copyright	
Date Generated:	06/10/2008	
Operating System: (including version number)	Microsoft Windows V.5.1	
Application Used: (including version number)	MATLAB V.7.4.0.287	
Media Type: (CDs, 3 1/2, 5 1/4 disks, etc.)	1 CD	
File Types: (.exe, .bat, .zip, etc.)	m	
Remarks: (computer runs, etc.)		