Application of Discrete Event Simulation to Modeling Reliability of Highly Parallel Systems with Common Cause Failures

by Scott Littlefield

B.S. in Mechanical Engineering, December 1982, University of Michigan
M.P.A., June 1991, Harvard University

A Praxis submitted to

The Faculty of
The School of Engineering and Applied Science
of the George Washington University
in partial fulfillment of the requirements
for the degree of Doctor of Engineering

January 31, 2017

Praxis directed by

Thomas Mazzuchi
Professor of Engineering Management and Systems Engineering

Shahram Sarkani
Professor of Engineering Management and Systems Engineering

The School of Engineering and Applied Science of The George Washington University certifies that Scott Littlefield has passed the Final Examination for the degree of Doctor of Engineering as of 1 November 2016. This is the final and approved form of the praxis.

Application of Discrete Event Simulation to Modeling Reliability of Highly Parallel Systems with Common Cause Failures

Scott Littlefield

Praxis Research Committee:

Thomas Mazzuchi, Professor of Engineering Management and Systems Engineering, Praxis Director

Shahram Sarkani, Professor of Engineering Management and Systems Engineering, Committee Member

Christopher Orlowski, Program Manager, Defense Advanced Research Projects Agency, Committee Member

Dedication

To my two wonderful daughters, Sarah and Rebecca.

Acknowledgements

Many people gave me encouragement and practical advice in the pursuit of this research, and throughout the course of this graduate program. I'd particularly like to thank Dr. William Roberts, whose course introduced me to the beauty and power of discrete event simulation. I'd like to thank my classmates, Dr. Daniel Deitz and Dr. Daniel Georgiadis, who car pooled to class with me and shared their experiences throughout the research phase of this program. Most of all, I'd like to thank Professor Thomas Mazzuchi and Professor Shahram Sarkani for their patience in what turned out to be a very long journey.

Abstract

Application of Discrete Event Simulation to Modeling Reliability of Highly Parallel Systems with Common Cause Failures

This praxis develops a simulation-based approach to analyzing the overall reliability of complex systems with high degrees of redundancy, time varying event rates, and the potential for common cause failures. This approach is compared to traditional analytic approaches, and is shown to have some advantages, primarily by avoiding some of the simplifying assumptions used in those approaches.

Several canonical problems are solved using both traditional and simulation-based approaches to elucidate the method, and the method is then applied to more complex problems for which exact analytic solutions are not available. The method is shown to be flexible to both traditional industrial plant reliability problems and to a new class of problems involving the reliability of swarming unmanned vehicles, where there is a high degree of parallelism and dynamic formation of common cause groups.

The penultimate chapter examines the impact of common cause failures on the reliability of a swarm of unmanned vehicles performing a search mission, and develops a simulation-based approach to modeling the reliability of swarms in the presence of both independent (single vehicle) and common cause (multiple vehicle) failures. The modeling approach is exercised on a sample problem to illustrate how it can be used as part of a system design or search-planning tool for swarming unmanned vehicles. The simulation provides insight on the impact of design decisions that influence overall system reliability; it also provides metrics of success in a search scenario as a function of user-selectable parameters.

Table of Contents

Dedication iii

Acknowledgments iv

Abstract v

List of Figures viii

List of Tables ix

Chapter 1: Introduction 1
1.1 Chapter Introduction 1
1.2 Motivation for Research 1
1.3. Limitations of Current Approaches 3
1.4 Discrete Event Simulation 6
1.5 Potential Applications 8
1.6 Outline 9

Chapter 2: Background and Literature Review 11
2.1 Chapter Introduction 11
2.2 Probabilistic Risk Assessment 11
2.3 Review of Some Concepts from Basic Reliability Theory 15
2.4 Independent and Dependent Failures; Common Cause Failures 17
2.5 Common Cause Failure Models 21
2.6 Best Practices and Procedures for Incorporating CCF Models into PRA 27
2.7 Failure Rate Distributions; Homogeneous and Non-Homogeneous Poisson Processes 29
2.8 Discrete Event Simulation 32
2.9 Discrete Event Simulation Process 35
2.10 Generation and Use of Random Variates to Produce Event Times in Simulation 36

Chapter 3: Discrete Event Simulation of Common Cause Failure – A Simple Example 39
3.1 Chapter Introduction 39
3.2 Traditional Analytic Approach 40
3.3 Discrete Event Simulation Approach, Step by Step 44
3.4 Discrete Event Simulation – Tying it all Together 50
3.5 Comparison of Simulation Result to Analytic Result 51

Chapter 4: Incorporating Failure on Demand 54
4.1 Chapter Introduction 54
4.2 Four-Component System Analytic Solution with Independent Components 54
4.3 Modeling and Simulation Approach 56
4.4 Simulation of Failure on Demand 58
4.5 Simulation of Run Time Failure 60
4.6 Complete Simulation of a Four-Component System 62
4.7 Simulation Output and Comparison to Analytic Results 65
4.8 Summary 68

Chapter 5: Incorporating Time-Varying Event Rates 69
5.1 Chapter Introduction 69
5.2 Thinning approach to incorporating time varying event rates 69
5.3 An Example of a k-out-of-n system with Weibull Failure Rate 71

Chapter 6: Auxiliary Feedwater System Example 76
6.1 Chapter Introduction 76
6.2 Description of the Auxiliary Feedwater System 76
6.3 Analytic Solution from the Literature 78
6.4 Comparison of Analytic and Simulation Solutions 79

Chapter 7: Application to a New Class of Problems – Swarming Unmanned Vehicles 82
7.1 Chapter Introduction 82
7.2 Dynamic Formation and Disbanding of Common Cause Groups 84
7.3 Development of a simulation model for the swarming vehicle problem 85
7.4 Simulation Parameters and Results 90
7.5 Chapter Summary 97

Chapter 8: Summary, Conclusions, and Future Research 99
8.1 Chapter Introduction 99
8.2 Usefulness of the Approach: Considerations of Scalability and Run Time 99
8.3 Summary and Conclusions 103

References 106

Appendix – MATLAB Code 111
Section A1- Auxiliary Feedwater System Simulation 111
Section A2 – Swarm Simulation 122

List of Figures

1	Simple example of application of series and parallel rules	16
2	Equivalent representations of a bridge network	17
3	Approach to modeling common cause failures	20
4	Example of NHPP behavior from a system made up of HPP components	30
5	Taxonomy of simulation approaches, adapted from Leemis and Park	33
6	A simple two-component parallel system	40
7	Fault tree for simple two component parallel system	42
8	Comparison of simulation result with analytic result for a two-component parallel system	52
9	Comparison of simulation result with analytic result for a two-component parallel system	52
10	Comparison of simulation result incorporating both start-up and run-time common cause failures to analytic result incorporating only independent run-time failures for a 2-out-of-4 system.	66
11	Close-up Comparison of Simulation Result with Analytic Result for a 2-out-of-4 system.	66
12	Comparison of Simulation Result incorporating both start-up and run-time common cause failures to Analytic Result incorporating only independent run-time failures for a 2-out-of-4 system.	68
13	Graphical representation of thinning technique	70
14	Simulation of Weibull Failure Distribution in Blue, analytic result for corresponding Exponential Failure Distribution in Red	75
15	Auxiliary Feedwater System studied by Fleming and Mosleh; numbers in circles correspond to component numbers in reliability block diagram	77
16	Reliability block diagram for auxiliary feedwater system	77
17	Survival function for auxiliary feedwater system over the first 100 hours of system operation.	80
18	Survival function for auxiliary feedwater system over five years of operation.	81
19	Vehicle locations in the search box at the time of failure.	85
20	Number of operable vehicles over time for a typical simulation run.	93
21	Number of operable vehicles over time averaged over 50 simulation runs.	94
22	Effect of common cause failures; curve gets closer to the origin as the common cause failure parameter Beta is increased.	95
23	Simulation run time as a function of number of random events generated	100
22	Simulation of “Bridge” system	102

List of Tables

1	Some Common Cause Failure Models	22
2	Process for conducting a Common Cause Failure Analysis	28
3	Some potentially useful failure distributions	32
4	Event probabilities with constant failure rates	38
5	Example result, constant event rates	38
6	Process for incorporating common cause failure analysis in a probabilistic risk assessment	39
7	Discrete Simulation Steps	44
8	Parameter values for Auxiliary Feedwater System	79

Chapter 1 – Overview and Motivation

1.1 Chapter Introduction

This chapter discusses the focus of the research undertaken in preparing this work, the motivations for the research, the relevance of the research to real problems, the general approach and rationale, and potential applications of the method developed.

1.2 Motivation for Research

A difficult issue in the design of any complex system is determining the number and configuration of redundant and back-up components and sub-systems needed to achieve desired levels of overall system safety, reliability, capacity, and graceful degradation. Redundant systems are frequently required by codes and standards in safety critical applications such as transport aircraft and commercial nuclear power plants. Even if not required by codes and standards, redundant and back-up systems are frequently employed as a way of achieving high levels of system availability; for example, the use of back-up diesel generators and redundant arrays of inexpensive discs (RAID) in data centers, where downtime or loss of data can have significant financial consequences.

Redundancy is also a design consideration in highly parallel systems, where the nature of the system naturally lends itself to use of many distributed components to achieve an objective; a recent example of this type of system is the use of many "swarming" unmanned vehicles to perform a sensing mission, where limitations of sensor range and vehicle endurance are overcome by use of many vehicles simultaneously to

rapidly perform a search mission. This system architecture naturally brings a degree of redundancy and graceful degradation, under the assumption that individual vehicles fail independently.

For the designer of a system, an obvious question is "how much redundancy is enough?" Redundancy costs money, both in the initial acquisition of a system and in subsequent maintenance and operations costs. In aerospace systems there are additional constraints of limited space, weight, and power. Designers need simple computational tools to predict the overall reliability of a system and to perform sensitivity analysis on alternative system configurations in order to achieve the best possible balance among competing design goals.

In the early days of reliability analysis, is was assumed that individual component failures were independent; this was a useful simplifying assumption that allowed the reliability of large interconnected systems to be analyzed through straightforward series and parallel rules, and enabled an analytic solution to overall system reliability based on the reliability of the individual components. However, as reliability theory advanced during the second half of the 20th century and was used in predicting the performance of real systems, it was observed that the assumption of independent component failures was frequently not valid, and that there were many examples of significant positive dependencies between the failures of nominally independent components. Therefore, reliability predictions based on the assumption of independence tended to over-value the benefit of redundant components and over-predict the resulting system reliability. In response to this observation, a number of common-cause failure models were developed to account for the actual dependence between nominally independent components, and a

significant body of work was published regarding the use of these dependent failure models to determine overall system reliability in the context of probabilistic risk assessment. This work is described in more detail in Chapter 2.

Probabilistic risk assessment has matured and advanced significantly over the last 40 years and has gained wide acceptance in design of safety critical systems. Standard practices for conducting these assessments incorporate common cause failure models to more accurately predict the reliability of the overall system. Current widely-used approaches to evaluating common cause failures invoke several standard models such as the Beta Factor and Binomial Failure Rate models. However, these commonly referenced models rely on simplifying assumptions that were initially invoked to make these analyses solvable, but which may not be adequate in some systems; these limitations are described in the next section.

1.3 Limitations of Current Approaches

There are three major limitations associated with the common cause failure modeling approaches described in current textbooks and handbooks: (1) Use of constant event rates in the model when actual event rates may not be constant, (2) Use of the rare-event approximation, and (3) The difficulty in applying these models in situations where the number and size of common cause groups is large, or where these groups may form and disband dynamically over time. Each of these limitations is described in more detail below.

1.3.1 Constant Failure Rate Assumption:

Many of the popular models invoked in common cause failure analysis use the simplifying assumption that pertinent events happen at a constant rate, i.e. a homogeneous Poisson process; more details on these current models are provided in Chapter 2. The constant event rate assumption ignores phenomena such as infant-mortality, aging, and other time-dependent effects. The use of models that don't account for time-varying failure rates has been identified as a limitation of current practice; therefore, development of time-varying models has been identified as a promising area for research. Hockstad and Rausand (2008) noted that:

> It is further observed that all current implicit models are based on the exponential failure model. The occurrence of CCF in models with deterioration (e.g. Weibull models) should be further investigated.

1.3.2 Rare Event Approximation:

Analysis of common cause failures using the existing models generally requires use of the "rare event" approximation, which states that the combined probability of two independent events is approximately the sum of their individual probabilities. For a system with two independent events, A and B, this is expressed as:

$$P(A \cup B) \approx P(A) + P(B)$$

The exact solution for two events is:

$$P(A \cup B) = P(A) + P(B) - P(A) \cdot P(B)$$

Given that both event A and event B are rare, the product of their probabilities is low, so the approximation is reasonable. Generalizing to systems in which there are multiple possible events, the rare event approximation is a truncation of the following exact solution for the probability of the union of multiple possible events (Collet, 1996):

$$P\left(\bigcup_{i=1}^{n} E_i\right) = \sum_{k=1}^{n} P(E_i) - \sum_{i<j\leq k} P(E_i \cap E_j) + \ldots + (-1)^{n-1} P\left(\bigcap_{i=1}^{n} E_i\right)$$

The rare event approximation truncates this after the first summation, forming an upper bound on the probability, while the inclusion of the second summation forms a lower bound; these are the Boole-Bonferroni bounds on probability of a union of events. It has been observed that the upper and lower bounds can in some cases be far enough apart that they are not sufficient to adequately bound the probability of system failure (Epstein and Rauzy, 2005). It can also be seen that extension of the exact solution to systems with more than a few possible events would result in an unwieldy number of terms, which is why the rare event approximation is used in many practical problems. If all of the events in the system are of low probability, then the terms after the first summation can be neglected with minimal error, providing a slightly conservative answer. However, to the extent that the events are not actually rare, significant error can be introduced. So for example, in a swarming vehicle application where the vehicles are expendable and intended to run to failure, the rare event approximation is not valid. More recently, binary decision diagrams (BDD) have been advocated as a method to avoid use of the rare event approximation, but with significant complexity in developing and reducing the BDD model; for many systems this can be either difficult or infeasible (Rauzy, 2008).

1.3.3 Large size and number of common cause groups; dynamic formation of groups:

Finally, with any significant number of vehicles in a swarm, the number of common cause groups can grow to intractable levels. In a typical industrial plant reliability problem, there might be several identical components arranged in parallel to

perform a common function. For example, a power plant might have four identical pumps, any two of which are needed for the plant to operate. In this system, there are potentially 15 unique failure events involving combinations of one or more of the pumps; five of these lead to overall system failure. In general the number of unique failures is bounded by $\sum_{r=1}^{n} \binom{n}{r}$, where *n* is the number of components in a common cause group. For a swarming vehicle application, where *n* may be a very large number of vehicles, it becomes computationally intractable to explicitly enumerate all of the different combinations of failures. Furthermore, if the common cause failure is due to environmental phenomena that are limited in geographic extent, or due to proximity among vehicles, then it may be necessary to dynamically form and disband common cause groups as individual vehicles move in and out of that area.

1.4 Discrete Event Simulation

Discrete event simulation is a powerful tool for evaluating the behavior over time of systems for which closed-form analytic solutions aren't achievable or would be excessively complex. Discrete event simulation can readily accommodate realistic assumptions about changing component failure rates and other event rates, and is scalable to complex systems involving multiple components and alternate modes of operation. Discrete event simulation models, properly implemented, can also be used to conduct sensitivity analysis, providing useful design insights in early stages, when the actual levels of component reliability and dependency are uncertain.

The focus of this work is on development of a flexible discrete event simulation approach for conducting probabilistic risk assessment of systems where both independent

component failure rates and common-cause event rates are allowed to change over time, and where common cause groups can form and disband dynamically over time. Although a number of specialized simulation software packages are available, the current research implements these common-cause failure models through discrete event simulation in MATLAB. Use of MATLAB provides transparency of the algorithms being implemented, aiding in verification of the models. MATLAB has the additional advantage of being widely used in engineering, so engineers who are not programmers can readily learn and apply the methods developed herein, and can build their own models without buying specialized software. Complete MATLAB code with annotations is included herein for each of the example problems discussed, to aid the reader in reproducing or extending the modeling method. The basic approach developed in this research could also be implemented in a general-purpose language such as C or FORTRAN.

It will be shown that the discrete event simulation method can be implemented within the context of existing processes for conducting a probabilistic risk assessment, that the methods are scalable so as to be useful for real systems, that the models developed are relatively compact and run quickly on standard computers, and that the method, once understood, is practical and relatively easy to learn.

1.5 Potential Applications

The inclusion of time-varying event rates in common cause failure models is potentially useful in a wide variety of applications. As one example, in the design of a deep space probe, environmental conditions such as intensity of solar radiation will

change over time. Batteries and other energy storage systems will age. Components will fail and can't be repaired during flight. This may be a domain in which the incorporation of time-varying failure rates would lead to more accurate predictions of lifetime and therefore better assessments of probability of mission success.

In highly-maintained systems such as commercial power or chemical process plants, it may be argued that time-varying event rates are less important, if maintenance plans are configured to keep the plant in peak condition throughout its life, and if components subject to aging are replaced well before the time of expected failure. However, many components in power and chemical plants that are designed to last over the entire life of the plant actually do exhibit aging phenomena, such as erosion, corrosion, embrittlement, and fatigue. Inclusion of these aging phenomena in plant probabilistic risk assessment models might provide greater insight into decisions about when to shut a plant down, whether or not to re-license a plant, and when to change inspection periodicity to detect aging phenomena.

Discrete event simulation provides the flexibility to treat the individual component failure rates and common cause event rates separately, based on the underlying physical phenomena that lead to each type of failure. For example, if it is determined that individual component failures are largely caused by aging, but that common cause events are largely driven by human error or external phenomena such as earthquakes, it would be reasonable to model the individual component failure rates using a time varying approach such as the Weibull distribution, while modeling the common cause events with an Exponential distribution. This is easily done using the methods described herein.

An emerging application which is addressed herein is the use of discrete event simulation to model common cause failures in a swarm of small unmanned vehicles. A swarm is a highly parallel system that is intentionally designed to maximize the independence of the individual vehicles by avoiding hierarchical command and control structures. It is nevertheless vulnerable to common cause failures, which may be due to common flaws in hardware or software design, or to common environmental conditions that multiple vehicles are exposed to. One interesting nuance of this application is the concept that common cause groups may form and disband in a dynamic fashion based on temporal clustering of the vehicles near each other or in a particular geographic region.

1.6 Outline

The remainder of this work is organized as follows:

- **Chapter 2** provides a review of some basic concepts of reliability theory that are necessary to develop the simulation method, a short summary of the history of probabilistic risk assessment with emphasis on the development of common cause failure models and related analysis procedures, and a brief introduction to discrete event simulation.
- **Chapter 3** provides a simple example of discrete event simulation of reliability of a two-component system and a step-by-step illustration of the simulation process. The simulation results of this example are compared to the exact analytic solution.
- **Chapter 4** provides a somewhat more complex example that incorporates failure on demand and begins to show some of the advantages of the discrete event simulation for analysis of a system that includes common cause failures.

- **Chapter 5** continues the analysis of the sample problem introduced in Chapter 4, adding the complexity of time-varying failure rates.
- **Chapter 6** explores the use of discrete event simulation in a realistic industrial plant application which has been studied extensively in the literature, and compares the simulation result to the published solution. This chapter shows that the rare event approximation can be significantly wrong in cases where this approximation is not appropriately used. Discrete event simulation is shown to have an advantage in this case.
- **Chapter 7** extends the simulation method to a highly parallel system of swarming unmanned vehicles and shows the flexibility of the model to this class of problem, which is significantly different than typical industrial plant reliability problems.
- **Chapter 8** addresses issues of scalability and run time and provides an overall summary of the work.

Chapter 2: Background and Literature Review

2.1 Chapter Introduction

This chapter provides a brief history and literature survey of the development of probabilistic risk assessment (PRA) as a preferred method for evaluating the reliability of safety-critical systems, with a focus on common cause failure models and current practice for incorporating common cause failures into PRA. A brief review of some basic concepts from reliability theory is also provided, focusing on concepts and definitions that underlie common cause failure analysis and which provide a foundation for the methods developed in later chapters. This chapter also describes the development of discrete event simulation as a method for solving a wide variety of problems that don't lend themselves to closed form solutions, and discusses general applicability of the discrete event simulation method to the set of problems that are the focus of this research.

2.2 Probabilistic Risk Assessment

The human desire to quantify risk has its origin in ancient times and arose in economic domains such as gambling, insurance, and investment. However, the mathematical foundations that allowed problems of engineering risk to be evaluated quantitatively did not come into being until much later, following the work of Pascal, Fermat, and others (David, 1962). Andrey Kolmogorov codified the framework for modern approaches to probability theory in the 1930s, and most modern texts in probability are based on his work (Kolmogorov, 1950; Gerovitch, 2013). In addition to the lack of mathematical tools to quantitatively assess complex problems of engineering

risk, many of these problems are of a size and complexity that can only be practically solved with computers. For these and other reasons, early methods of engineering design did not rely on mathematical quantification of risk using probability theory, but instead evolved using empirical and deterministic approaches, including the use of safety factors to accommodate unknown risks and minimize the probability of catastrophic failure (Beal, 2011).

The use of probabilistic methods to assess engineering risk began shortly after World War I, with application to evaluating the safety of multi-engine aircraft, but these methods didn't reach maturity until the later portion of the last century. Motivated by difficult problems of system reliability in the military and aerospace domain and nuclear power safety in the civil domain, formal methods and tools of reliability theory such as fault tree analysis and probabilistic risk assessment (PRA) began to emerge (Rausand and Hoyland, 2004; Ericson, 1999).

A major milestone in the emergence of probabilistic risk assessment was the conduct of a comprehensive study of commercial nuclear power safety, known variously as the Reactor Safety Study, WASH-1400, or the "Rasmussen Study" (Rasmussen, et. al., 1975). This study was, at the time, the most comprehensive assessment of the risk of a major release of radioactive material from a commercial nuclear power plant accident. The Reactor Safety Study led to significant changes in the design philosophy and regulatory framework for the commercial nuclear power industry in the United States. Prior to 1975, plant safety relied primarily on an approach called "defense in depth" which relied on adequate factors of safety and degrees of redundancy for critical systems to guard against failure. After 1975, nuclear regulators in the United States and

elsewhere began to shift toward approaches that use probabilistic assessment of risk. Although both methods have utility in design of an adequately safe system, the probabilistic methods are more useful in determining where and how to focus resources to improve safety, and determining where additional resources provide little or no benefit (Keller and Modarres, 2005; Apostolakis, 2004).

Probabilistic risk assessment has its critics, particularly among those who advocate against the commercial use of nuclear power. Because of its prominence in informing the nuclear safety debate in the 1970s, the greatest period of growth in the U.S. nuclear power industry, the Reactor Safety Study became a political document, used by nuclear power supporters as part of the argument for asserting that nuclear power is safe, and attacked by detractors, who pointed out perceived weaknesses in the report's analysis (Carlisle, 1997). Although widely adopted in engineering practice, the PRA method is hard to explain to the general public. As noted by Carlisle (1997):

> The failure to communicate the industry's own methods may account, at least in part, for the continuing public distrust of nuclear reactors devoted to the peaceful generation of electrical power. To describe those methods, however, would require mounting a widespread effort to build understanding and trust in the mathematics of probability.

Both critics and supporters of PRA have noted some of the limitations of the method, including the difficulty of imagining and exhaustively analyzing all possible failure modes and scenarios, and the lack of relevant quantitative data from which to determine the individual component failure rates and other event rates upon which an overall assessment of system risk is based. Sociologists have criticized PRA for failing to pay adequate attention to human and organizational factors both in predicting the behavior of plant operators and maintainers, and in evaluating the opinions of experts

who are called upon to provide opinions or analysis that feed into the PRA (Freudenberg, 1988). One of the key figures in the development of PRA, George Apostolakis, acknowledges some limitations of quantitative risk assessment, including human error during accident conditions, failure of digital software, presence or absence of safety culture, and design and manufacturing errors (Apostolakis, 2004).

Notwithstanding the critics, the probabilistic risk assessment methodology developed in the Reactor Safety Study and refined subsequently has taken strong root in the design process for an increasing number of safety-critical systems, and has arguably been successful in improving the performance of some safety critical systems, such as commercial nuclear power plants and commercial aviation. Current policy of the United States Nuclear Regulatory Commission (NRC) encourages use of Probabilistic Risk Assessment as part of a "risk-informed" approach to regulation (NRC, 2007), and use of these methods has also met with wide acceptance by the commercial nuclear industry (EPRI, 2008). Although most of the commercial nuclear power plants in operation today were designed prior to the widespread adoption of PRA methods, PRA methods are integral to the NRC's process for licensing of new plants (NRC, 2009). A recent study by the General Accounting Office (GAO, 2012) motivated by the disaster at Fukushima encourages the increased use of PRA methods to assess the risk to commercial nuclear power plants from natural hazards. Probabilistic risk assessment methods are also being applied more widely in other technical domains including aviation and space applications. The National Aeronautics and Space Administration (NASA) has embraced PRA as a requirement for certain safety-critical missions (NASA, 2011).

2.3 Review of Some Concepts from Basic Reliability Theory

There are several basic concepts and definitions from reliability theory that will be central to the development of the discrete event simulation method; a brief review of those concepts is in order. First, assume that the system being modeled consists of *n* components, and that at any given time each component is in one of two states, either operable or failed. For each component there is an indicator variable x_i equal to one if the component is operable and zero otherwise. The vector $\mathbf{x} = \{x_1, x_2, \ldots x_n\}$ is then called the *state vector* for the system. Further assume that the overall system can also only be in two states, operable or failed, and that the system state is entirely determined by the states of the individual components as represented by the state vector. In this case, we can define a function $\phi(\mathbf{x})$, called the *structure function*, which equals one when the system is operable, and zero when the system is not operable (Ross, 1993).

Given the definitions above, a system of *n* components can be defined as a combination of series and parallel structures with the following definitions (Ross, 1993):

$$\text{Series structure:} \qquad \phi(\mathbf{x}) = \min(x_i \ldots x_n) = \prod_{i=1}^{n} x_i$$

$$\text{Parallel structure:} \qquad \phi(\mathbf{x}) = \max(x_i \ldots x_n) = 1 - \prod_{i=1}^{n} (1 - x_i)$$

A pictorial representation of the components of a system as a network of series and parallel components is called a reliability block diagram (Rausand and Hoyland, 2004), and the series and parallel rules can be used to determine the overall reliability of the system given the reliability of the components. A more detailed definition of reliability is given below.

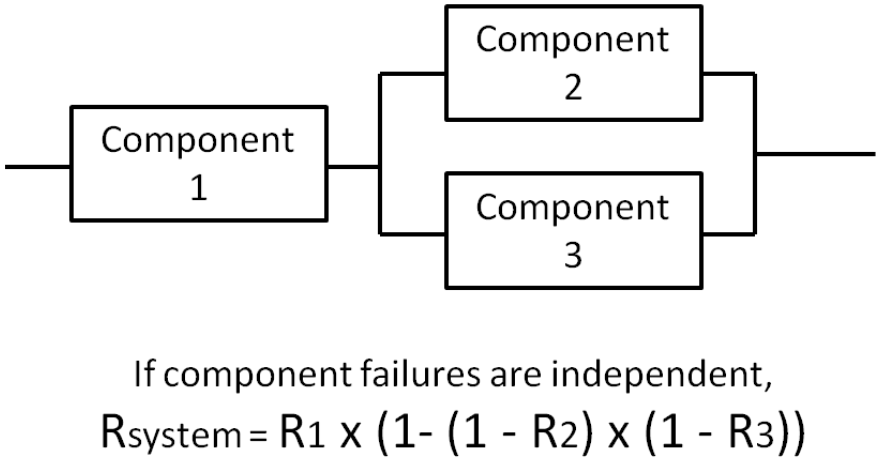
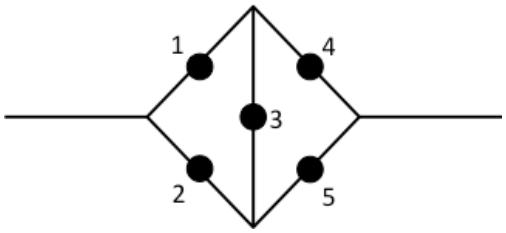


Figure 1 – Simple example of application of series and parallel rules

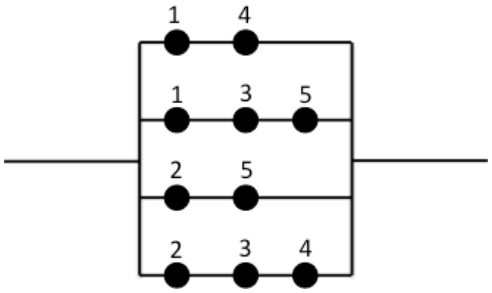
Note that even systems that don't obviously lend themselves to depiction as a network of series and parallel elements can generally be decomposed into such a network. Consider for example a "bridge system" as shown in figure 2, part (a). This system can also be represented by the equivalent series and parallel networks shown in figure 2, part (b) and (c) (Ross, 1993).

Given the structure function of a system and information about the reliability of the individual components in the system, it is generally possible to develop a mathematical expression for the reliability of the system, although some simplifying assumptions may be needed to make the mathematics tractable. For purposes of this discussion, reliability of a component or system is defined as "the ability of an item to perform a required function under given environmental and operational conditions and for a stated period of time (Rausand and Hoyland, 2004)."

(a) Bridge Network:



(b) An equivalent representation:



(c) Another equivalent representation:

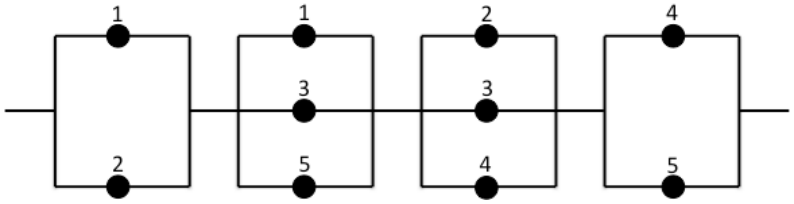


Figure 2 – Equivalent representations of a bridge network (Ross, 1993)

2.4 Independent and Dependent Failures; Common Cause Failures

A simplifying assumption in the early development of reliability theory was that failures of individual components are independent. This assumption enables the use of straightforward series and parallel rules to develop an analytical solution to overall system reliability based on the reliabilities of the individual components of the system (Rausand and Hoyland, 2004). Independence is defined as follows:

$$Pr(A_1 \cap A_2) = Pr(A_1) \bullet Pr(A_2)$$

Suppose that A_1 is the event that component #1 fails and A_2 is the event that component #2 fails. $Pr(A_1)$ and $Pr(A_2)$ are the individual probabilities of failure in a specified time period. The reliability of component A is 1 – Pr(A). If A_1 and A_2 are independent and if the individual components are both reasonably reliable, then the product $Pr(A_1) \bullet Pr(A_2)$ will be a small number; it can be seen in this case that redundancy is an effective method of achieving high levels of reliability.

Unfortunately, experience has shown that the assumption of independent component failures is frequently wrong, and can lead to excessive confidence in redundancy as a solution to reliability problems. This was particularly noticeable in the early experience with commercial nuclear power plants, where unanticipated dependencies were noted to be a significant source of operational failures (Fleming, et. al., 1986). In highly redundant systems, where the probability of near-simultaneous independent failures is vanishingly small, dependent failures can be responsible for the majority of remaining reliability concerns. NASA identifies several examples of common cause failures leading to major catastrophes, most famously the 1986 CHALLENGER accident, where the primary and back-up O-rings that sealed the segments of the solid rocket boosters were assumed to fail independently, but actually failed due to a common environmental cause, the cold weather on the morning of the launch (NASA, 2011).

Since dependent failures were identified as a significant concern in reliability theory, a body of academic work developed to categorize and define various types of dependent failures. The general definition of dependence is the following:

$$Pr(A_1 \cap A_2) \neq Pr(A_1) \bullet Pr(A_2)$$

Note that the case where $Pr(A_1 \cap A_2) > Pr(A_1) \bullet Pr(A_2)$ is called positive dependence, and the opposite case, $Pr(A_1 \cap A_2) < Pr(A_1) \bullet Pr(A_2)$, is called negative dependence.

Rausand and Hoyland identify three types of dependent failures (2004):

a. Common cause failures, in which two or more failure events occur nearly simultaneously and due to a shared cause. This type of dependent failure is the primary focus herein, and will be defined in greater detail below.

b. Cascading failures, in which the failure of one component initiates the failure of other components.

c. Negative dependencies, in which a single failure reduces the probability of failure for other related components.

Kumamoto and Henley use a slightly different taxonomy (1996):

a. Functional and common-unit dependency, which includes cascade failures, propagating failures, or command failures.

b. Common-cause failures.

c. Subtle dependency, which includes standby redundancies, common loads, and mutually exclusive events.

In general, positive dependencies are of greater concern in reliability modeling than negative dependencies. Ignoring negative dependencies is generally a conservative assumption. Another way of grouping dependent failures is by whether or not they are amenable to explicit modeling. Kumamoto and Henley (1996) indicate that functional and common-unit dependencies are generally amenable to explicit modeling, while common cause failures represent the residual cases that must be treated through implicit

models described below. Some types of events such as fires and floods aren't "components" of the system being analyzed, but are potential root causes of dependent failures. The analyst may make a judgment call to treat these events implicitly within a common cause model or to explicitly model them as "pseudo-components." Both references agree that dependent failures that can be explicitly modeled should be modeled that way. Common cause failure models therefore typically deal with the residual dependencies that can't readily be modeled explicitly (Kumamoto and Henley, 1996; Rausand and Hoyland, 2004). This taxonomy and approach to modeling is shown in Figure 3:

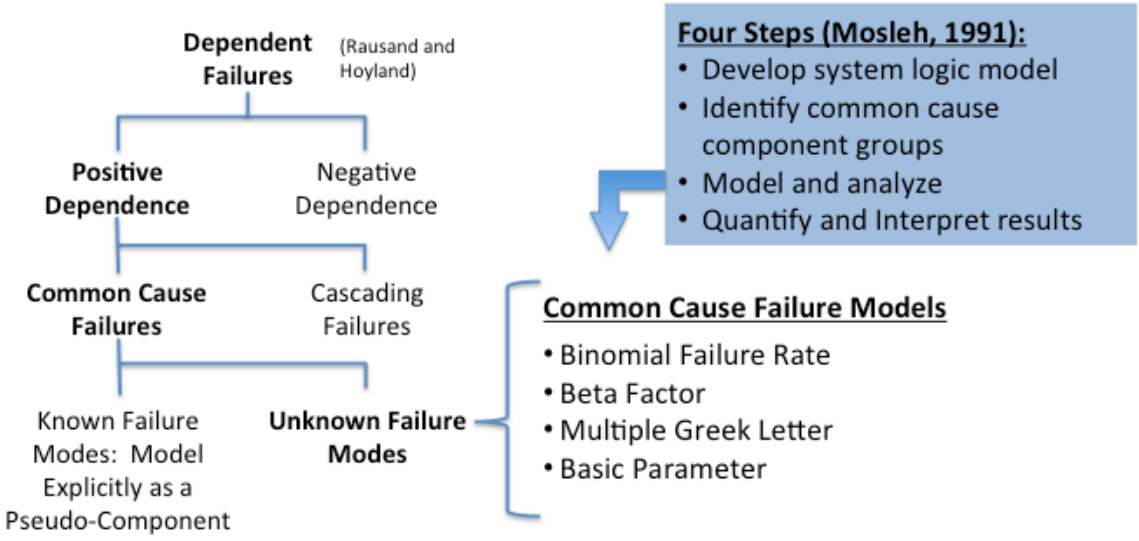


Figure 3 – Approach to modeling common cause failures

Smith and Watson (1980) reviewed the early literature on Common Cause Failures and identified multiple definitions of CCF ranging from expansive to narrow. They proposed the following definition:

> Inability of multiple, first-in-line items to perform as required in a defined critical time period due to a single underlying defect or physical phenomena such that the end effect is judged to be a loss of one or more systems. (Smith and Watson, 1980).

Another definition states that:

> . . . common cause events area a subset of dependent events in which two or more component fault states exist at the same time, or in a short time interval, and are a direct result of a shared cause. It is also implied the shared cause is not another component state because such cascading of component states is normally due to a functional coupling mechanism. (Mosleh, et. al., 1989)

Along with definitions, researchers developed several analytic models of common cause failure and a set of best practices and procedures to incorporate these models into the probabilistic risk assessment process, described in the following sections.

2.5 Common Cause Failure Models

The analysis of common cause failures employs several popular models, as shown in Table 1. This is not an exhaustive list, but includes the most commonly used models. The origins and particulars of these models are described in greater detail below.

Model	Reference	Description
Beta-Factor Model	Fleming, 1975	CCF events occur at rate $\lambda_c = \beta\lambda$, causing all components to fail simultaneously. Individual components fail separately at rate λ_i. The total failure rate for a single component is $\lambda = \lambda_i + \lambda_c$
Multiple Greek Letter Model	Fleming, 1986	Given a failure in a multi-component system, failure probabilities are distributed as: Pr (One component fails): $1 - \beta Q$ Pr (Two components fail): $\beta (1 - \gamma) Q$ Pr (Three components fail): $\beta\gamma Q$ [Extended to four or more components by adding additional Greek letters]
Binomial Failure Rate Model	Marshall & Olkin, 1967; Vesely, 1977; Atwood, 1986	CCF events occur at rate ν, causing individual components to fail with probability p. The number of components that fail is binomially distributed. The total failure rate for a single component is $\lambda = \lambda_i + \nu p$
Basic Parameter Model	Apostolakis et. al., 1986	CCF events occur at rate λ within a common-cause component group with m components. Given that an event has occurred, Q_k is the probability that exactly k out of m components failed in that event. This is the most flexible model, but requires the greatest number of parameters.

Table 1 – Some Common Cause Failure Models

One way of describing common cause failure is through the concept of "shock" events that cause the simultaneous failure of one or more components in a system. Several of the common cause models can be developed starting from this concept. Following the work of Marshall and Olkin (1967), assume that a two-component system has three associated independent Poisson processes:

$$Z_1 (t, \lambda_1); Z_2(t, \lambda 2); \; Z_{12}(t, \lambda_{12});$$

Events in the process Z_1 are shocks to the first component, events in the process Z_2 are shocks to the second component, and events in the process Z_{12} are shocks to both components. The parameter λ is the rate at which each type of shock occurs, following

an exponential distribution. The total event rate can be defined as the sum of the individual event rates:

$$\lambda_T = \lambda_1 + \lambda_2 + \lambda_{12}$$

Shocks can be further specified as fatal or non-fatal. A fatal shock always causes the failure of the components associated with that shock, while a non-fatal shock produces failures within the associated components according to a defined probability distribution.

Although the Marshall and Olkin formulation is a general shock model formulation, and seems relatively manageable in the two-component formulation shown above, it becomes unwieldy in situations with larger numbers of components. For a system with n components, the formulation requires up to 2^n-1 unique parameters to describe all of the possible event rates (Li, 2005). For example, with a common cause component group consisting of four components, the following fifteen parameters would be needed to fully describe the possible event rates, assuming no assumptions of symmetry: (λ_1, λ_2, λ_3, λ_4, λ_{12}, λ_{13}, λ_{14}, λ_{23}, λ_{24}, λ_{34}, λ_{123}, λ_{124}, λ_{134}, λ_{234}, λ_{1234}). Therefore, less general but less complicated models requiring fewer parameters were developed, and are described below. The selection of parameters for these models is motivated both by the ability to derive the model parameters from available data, and by the simplicity of the mathematics needed to determine an overall failure rate or probability from the model (Kumamoto and Henley, 1996; Rausand and Hoyland, 2004). In the development of the discrete event simulation method in later chapters it will be useful to be able to translate among these models, since in many cases a model that is chosen for convenience of determining parameters or developing an analytic solution may not be the easiest model for which to develop simulation code.

2.5.1 Basic Parameter Model

The Basic Parameter model extends the Marshall-Olkin model to include not only time-based but also demand-based failure probabilities. A demand-based failure probability is the finite probability that a component will not function when demanded by a change of state of the system. An example would be failure of a backup diesel generator to start in response to an automatic call from a control system. The Basic Parameter model also assumes symmetry among nominally identical components, which can substantially reduce the number of required parameters (Apostolakis, et. al., 1986; Kumamoto and Henley, 1996). For a common cause group of n standby components which must start and continue to run for time t, there are $2n + 1$ parameters of the following form (Kumamoto and Henley 1996):

λ_{sj} = failure probability of a particular group of j components to start on demand

λ_{rj} = failure rate (per unit time) of a particular group of j components..

t = mission time

In the example given above of four identical components, there are nine parameters: λ_{s1}, λ_{s2}, λ_{s3}, λ_{s4}, λ_{r1}, λ_{r2}, λ_{r3}, λ_{r4}, and t. Note that some of the parameters apply to multiple groups; for example, in the four-component system there are six possible groups of two components, so the probability of two components failing on demand is 6 λ_{s2}.

2.5.2 The Beta Factor Model

One of the simplest and most commonly used models is the Beta Factor model, originally proposed by Fleming (1975). This model assumes that all of the individual components in a common cause group are identical, and therefore the individual

component failure rates reduce to a single value λ_i. Additionally, the model assumes that the only common cause event is a lethal shock to all of the components in the group, λ_c. So in comparison to the Marshall-Olkin formulation above, the fifteen parameters needed to describe a four-component group reduce to three parameters:

$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda_i$$

$$\lambda_{12} = \lambda_{13} = \lambda_{14} = \lambda_{23} = \lambda_{24} = \lambda_{34} = \lambda_{123} = \lambda_{124} = \lambda_{134} = \lambda_{234} = 0$$

$$\lambda_{1234} = \lambda_c$$

The total failure rate for an individual component is:

$$\lambda = \lambda_i + \lambda_c$$

The common cause event parameter λ_c is typically expressed as a proportion of the total failure rate, β, explaining why it is referred to as the Beta Factor Model:

$$\lambda_c = \beta\lambda$$

The assumption that any common cause event will produce a lethal shock to all components in the component group may be an unduly conservative assumption, since it doesn't accurately model the redundancy benefit obtained by having more then two components in a parallel configuration (Rausand and Hoyland, 2004).

2.5.3 The Multiple Greek Letter Model

The Multiple Greek Letter Model is an extension of the Beta Factor model to groups of greater than two components, allowing for common cause failures of more than one but less than all of the components. It incorporates both failure on demand and time based failures. For a system of three components (Hockstad and Rausand, 2008):

Q = Total probability of any failure in the group

1 - βQ = Probability that the failure involves exactly one component

$\beta (1 - \gamma) Q$ = Probability that the failure involves exactly two components

$\beta\gamma Q$ = Probability that the failure involved exactly three components

This model can be extended to additional components by adding additional parameters.

2.5.4 The Binomial Failure Rate Model

The Binomial Failure Rate Model is a shock model, and a simplification of the Marshall-Olkin model (Rausand and Hoyland, 2004). In the original formulation by Vesely (1977) it allowed two types of failures: independent failures, which occur at rate λ, and failures due to a common cause shock event, which occur at rate ν. Unlike the Beta factor model, common cause failures aren't always considered fatal to all components; instead, the occurrence of a common cause event leads to failure of each individual component with probability "p." Therefore, the number of components that fail follows a binomial distribution:

$$\Pr (Z = z) = \binom{n}{z} p^z (1- p)^{n-z}, \text{ for } z = 0,1,2 \ldots$$

$$\text{where } \binom{n}{z} = \frac{n!}{z!(n-z)!}$$

The mean number of failed components is $E(Z) = np$ (Rausand and Hoyland, 2004). An extension of the Binomial model proposed by Atwood (1986) allows for third type of event, which is a lethal shock that fails all of the components in the group. Adding this possibility, for a four-component group, the individual and common cause failure parameters are:

a) Failure of a single component either individually or by common cause:

$$\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = \lambda + 4p(1-p)^3\nu$$

b) Failure of two components by common cause (non-lethal shock):

$$\lambda_{12} = \lambda_{13} = \lambda_{14} = \lambda_{23} = \lambda_{24} = \lambda_{34} = 12p^2(1\text{-}p)^2\nu$$

c) Failure of three components by common cause (non-lethal shock):

$$\lambda_{123} = \lambda_{124} = \lambda_{134} = \lambda_{234} = 4p^3(1\text{-}p)\nu$$

d) Failure of all four components by common cause (non-lethal or lethal shock)

$$\lambda_{1234} = p^4\nu + \lambda_f$$

2.6 Best Practices and Procedures for Incorporating CCF Models into PRA

Following the issuance of the Rasmussen Report, the Nuclear Regulatory Commission sponsored work to develop standard processes for conduct of Probabilistic Risk Assessment, including incorporation of common cause failure models. The work to develop standard processes for common cause failure analysis was documented in several reports and journal articles, most notably including work by Karl Fleming, Ali Mosleh and others (Fleming, et. al., 1986; Mosleh, et. al., 1989; Mosleh, 1991). A more recent survey by Hockstad and Rausand (2008) provides an evolved description of this process. A comparison of the analysis processes as outlined in each of these references is provided in Table 2:

<table>
<tr><td>Fleming, et. al., 1986</td><td>Mosleh, et. al., 1989; Mosleh, 1991</td><td>Hockstad and Rausand, 2008</td></tr>
<tr><td></td><td rowspan="3">System Logic Model Development
• System Familiarization
• Problem Definition
• Logic Model Development</td><td rowspan="3">Development of system logic models</td></tr>
<tr><td>System Familiarization</td></tr>
<tr><td></td></tr>
<tr><td rowspan="4">Logic Model Development</td><td rowspan="5">Identification of Common Cause Component Groups
• Qualitative Analysis
• Quantitative Screening</td><td>Identification of Common Cause Component Groups</td></tr>
<tr><td>Identification of root causes and coupling factors</td></tr>
<tr><td>Assessment of component defenses</td></tr>
<tr><td>Explicit modeling</td></tr>
<tr><td>Boolean Analysis</td><td rowspan="4">Implicit modeling</td></tr>
<tr><td></td></tr>
<tr><td>Algebraic Model Development</td><td rowspan="2">Common Cause Modeling and Data Analysis
• Definition of Common Cause Basic Events
• Selection of Probability Models
• Data Classification and Screening
• Parameter Estimation</td></tr>
<tr><td>Parameter Estimation</td></tr>
<tr><td>System Quantification</td><td>System Quantification and Interpretation of Results
• Quantification
• Results Evaluation and Sensitivity Analysis
• Reporting</td><td>Quantification and interpretation of results</td></tr>
</table>

Table 2 – Process for conducting a Common Cause Failure Analysis

2.7 Failure Rate Distributions; Homogeneous and Non-Homogeneous Poisson Processes

A homogeneous Poisson process is a counting process with constant rate λ for $t \geq 0$. The process has independent increments, and the number of events in an interval of length t is Poisson distributed with mean λt. This is also known as a stationary process (Rausand and Hoyland, 2004; Ross, 1993). The exponential distribution used in the current set of widely used common cause failure models shown in Table 1 is a homogeneous Poisson process. The use of the exponential distribution in common cause failure analysis frequently allows an analytic solution for system reliability, and may allow the overall reliability of a system to be expressed as a simple number, for example 10^{-6} failures per year, independent of any aging or other process that may change the reliability of a system over time. For systems where data is sparse, or where the underlying physics supports the assumption of a constant failure rate, there is rationale for using an exponential distribution to represent component failures. However, the fact that all of the current widely used common cause failure models found in textbooks and handbooks require use of a constant failure rate is unduly constraining, given that some basic event rates in some systems are likely better represented by other distributions that allow failure rates to either increase or decrease over time.

The non-homogeneous Poisson process, in contrast is a process where the failure rate is a function of time, $\lambda(t)$. The process still has independent increments, and the number of events in an interval t is still Poisson distributed. The mean number of failures in an interval (v, t+v] is (Rausand and Hoyland, 2004):

$$\int_{v}^{t+v} \lambda(u)\,du$$

It is also worth noting that non-stationary behavior arises even in systems where all the individual components have constant failure rates. A simple example of this is a parallel system of two redundant independent components, each with a constant failure rate λ. The reliability of the system is:

$$R_{sys} = 1 - (1-R_1)(1-R_2)$$

$$R_{sys} = 1 - (1-e^{-\lambda t})^2 = 2e^{-\lambda t} - e^{-2\lambda t}$$

To show that this is not a homogeneous Poisson process, Figure 4 plots the system reliability (blue line) against component reliability for $\lambda = 0.01$ (red line), and against an HPP system that has equivalent reliability to the NHPP system at t = 100 (green line).

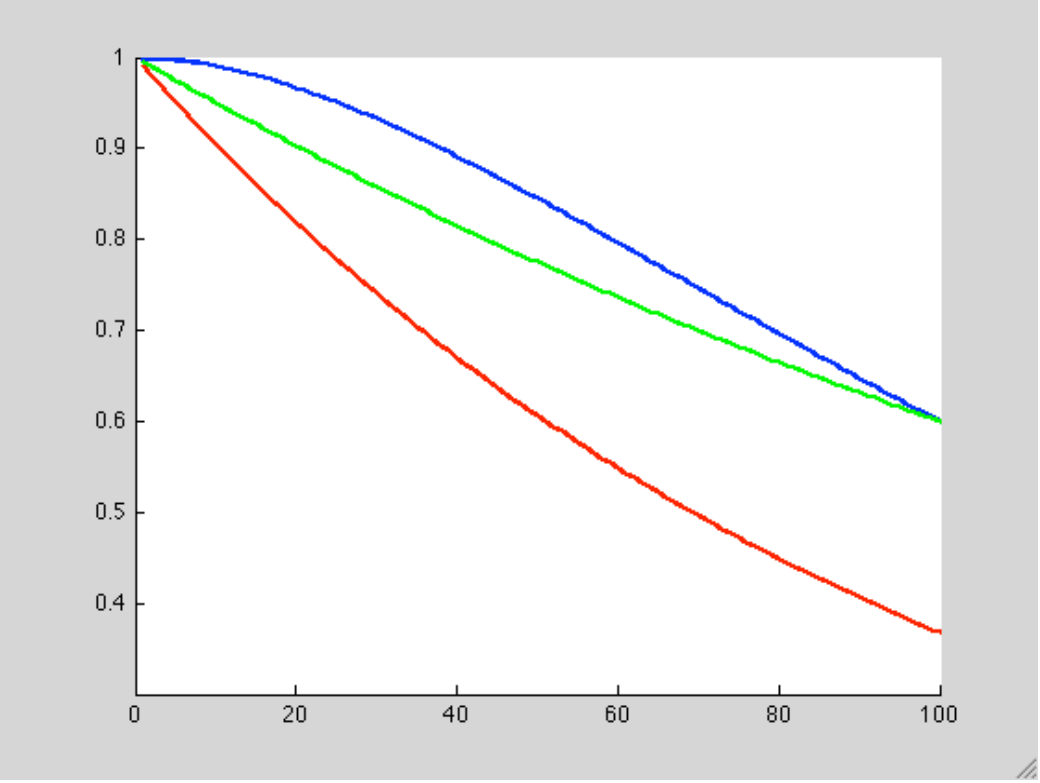


Figure 4 – Example of NHPP behavior from a system made up of two HPP components.

The plot was generated using the following MATLAB code:

```
t=0;
lambda = 0.01;
lambda2 = 0.0051;
R = zeros(1,100);
R2 = zeros(1,100);
R3 = zeros(1,100);
while t<100
    t = t+1;
    R(t) = exp(-1*lambda*t);
    R2(t) = 2*exp(-1*lambda*t)-exp(-2*lambda*t);
    R3(t) = exp(-1*lambda2*t);
end
hold on
p=plot (R);
set(p,'Color','red','LineWidth',1.05)
p=plot (R2);
set(p,'Color','blue','LineWidth',1.05)
p=plot (R3);
set(p,'Color','green','LineWidth',1.05)
```

Several alternatives to the exponential failure rate distributions are the Weibull, Gamma, Normal, and Lognormal distributions. Each of these distributions is defined in terms of a reliability or survivor function and a failure rate function, where the failure rate changes over time. The reliability or survivor function is defined as:

$$R(t) = \Pr(T>t) \text{ for } t>0$$

Where T is the time of failure. In other words, R(t) is the probability that the system or component will not fail in the time interval (0,t]. The failure rate function is defined as:

$$z(t) = \lim_{\Delta t \to 0} \frac{\Pr(t < T \leq t + \Delta T | T > t}{\Delta t}$$

In other words, *z(t)* is the instantaneous failure rate at time *t.* (Rausand and Hoyland, 2004). In a constant rate process, the failure rate λ is not a function of time. Table 3 provides reliability and failure rate functions for some potentially useful failure distributions.

Distribution	Survivor Function	Failure Rate Function
Exponential	$e^{-\lambda t}$	λ
Gamma	$\sum_{n=0}^{k-1} \frac{(\lambda t)^n}{n!} e^{-\lambda t}$	$\frac{\lambda(\lambda t)^{k-1} e^{-kt}/\Gamma(k)}{\sum_n^{k-1}(\lambda t)^n e^{-\lambda t}/n!}$
Weibull	$e^{-(\lambda t)^{\alpha}}$	$\alpha\lambda^{\alpha} t^{\alpha-1}$
Lognormal	$\phi\left(\frac{v - \ln t}{\tau}\right)$	$\frac{\phi((v-\ln t)/\tau))/\tau t}{\phi((v-\ln t)/\tau)/\tau)}$

Table 3: Some potentially useful failure distributions (Rausand and Hoyland, 2004)

2.8 Discrete Event Simulation

The early history of modeling and simulation included a wide variety of activities such as the development of scaling laws for physical scale models of ships and aircraft and the use of analog computers to simulate physical phenomena. However, the kind of simulation we are discussing herein developed in tandem with the development of digital computers and software, and originated around the time of the Second World War, with the emergence of approaches such as the Monte Carlo technique for numerically solving very complicated integrals (Nance, 1993). A useful taxonomy of modeling and simulation technques is provied by Leemis and Park (2005) as shown in figure 5:

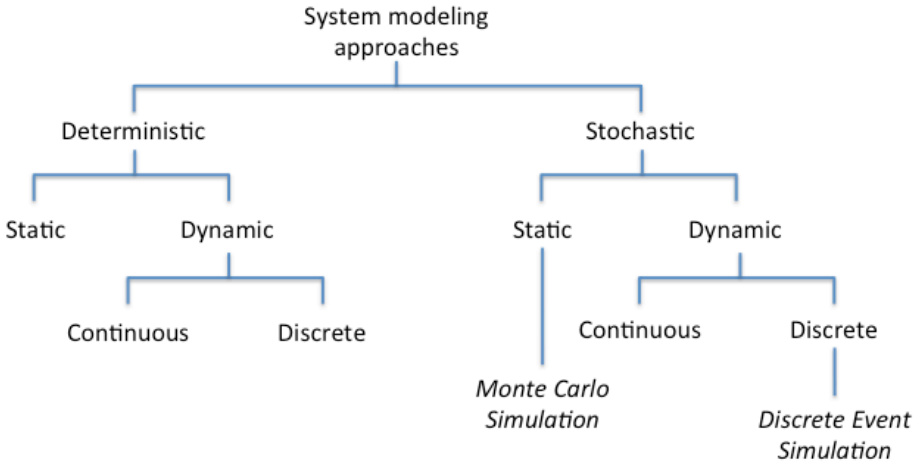


Figure 5 – Taxonomy of simulation approaches, adapted from Leemis and Park

Stochastic simulation involves systems where there is uncertainty about either the timing of an event or whether it will occur at all. Continuous simulation involves changes in state that do not result in discontinuities, for example water draining from a reservoir. Static modeling involves situations where explicit modeling of change over time is unnecessary. Therefore, discrete event simulation focuses on use of models that represent discontinuous state changes at specific but typically unknown points in time, and the evolution of system behavior or state over time in response to those changes. (Nance, 1993; Leemis and Park, 2005). Since reliability theory typically deals with events that are uncertain and discrete in time, for example the transition of a component from an operable to a failed state, discrete event simulation is an appropriate approach to Probabilistic Risk Assessment in general, and specifically to the analysis of common cause failures.

Although distinct from Monte Carlo simulation, discrete event simulation involves many of the same techniques and approaches, and the history of the two methods is thus intertwined. The Monte Carlo method was developed by Stanislaw Ulam

and John VonNeumann in the 1940s as a method to solve problems of neutron diffusion. Because the method required thousands of repetitive trials to generate meaningful statistics, it only became practical with the advent of digital computers (Metropolis and Ulam, 1949; Metropolis, 1987). A canonical problem in Monte Carlo Simulation is the solution of a definite integral by defining a shape in n-dimensional space, and then generating random points within a larger but solvable shape of known volume that contains the smaller shape. The ratio of points within the smaller shape to points in the larger shape gives the solution to the integral. A standard example of this type of problem is the solution of the value of Pi by defining random points within a square and measuring their distance from the origin (Roberts, 2010). Like discrete event simulation, the Monte Carlo method requires the generation of random numbers and the use of a statistically signficant number of trials to generate an answer that is reliably within specified error bounds.

The evolution of discrete event simulation as a useful tool for engineering and operations research is closely linked to the development of both general purpose programming languages and specialized simulation software packages, and the increasing availability of digital computers to the engineering community. Early computer programs were written in machine language, and were unique to the particular machines being used. By the 1960s, higher level languages were coming into use, and specialized simulation packages became available. One of the first discrete event simulation packages was General Purpose Simulation System (GPSS), which was widely used in the 1960s and 1970s. Other discrete event simulation packages have waxed and waned in popularity, but all have some commonalities, including the ability to generate random

events in time and to book-keep the evolving state of a system over time (Nance, 1993; Robinson, 2005).

It is also possible to implement discrete event simulation in a general purpose language such as FORTRAN or C. In the current context MATLAB is used, and for the purposes of taxonomy, MATLAB can be considered a general purpose language with some additional features that make is particularly convenient for development of discrete event simulation models, including easy handling of vectors and matrices and a built-in random number generator. Although use of MATLAB rather than a specialized simulation language adds some overhead to the process of developing a useful model, it aids in understanding the basic principles and provides a better view of the process "under the hood," and therefore is the approach taken in the present work.

2.9 Discrete Event Simulation Process

The discrete event simulation process includes the development of a simulation model, the running of the model, and the analysis and interpretation of simulation results. According to Leemis and Park (2005), the steps in the discrete event simulation process include the following:

- "Determine the goals and objectives of the analysis." In our case, the goal is to determine the reliability of a system over time, given some assumptions about the boundaries of the system, and the rates at which failure-causing events occur.
- "Build a conceptual model of the system." In our case, this might be a reliability block diagram. This step also includes definition of the state variables.
- "Convert the conceptual model into a *specification* model."

- "Turn the specification model into a computational model"
- "Verify the model"
- "Validate the model."
- "Design the simulation experiments"
- "Make production runs"
- "Analyze the simulation results"
- "Make decisions"
- "Document the results"

2.10 Generation and Use of Random Variates to Produce Event Times in Simulation:

The process for simulating the random time of occurrence of a component failure typically begins with a software routine that generates a random variate from the uniform distribution on the interval [0,1]. Note that in reality these routines generate "pseudo-random" numbers that are fully specified by a seed and by the algorithm used to generate the random number (Leemis, 2010; Roberts, 2010). From a practical standpoint, this doesn't matter, as long as the random variate behaves like a random number from a statistical point of view. There is a large body of literature and textbook chapters devoted to the topic of generating random variates (Law and Kelton, 1991; Leemis and Park, 2005). Within MATLAB, the command "rand" returns a pseudo-random number from the uniform distribution (Mathworks, 2016). The algorithm used within MATLAB to generate random numbers has evolved over versions of the software; in the version used for most of this work (Version 2012a) the algorithm used to generate random numbers is the "Mersenne Twister" (Matsumoto and Nishimura, 1998). This generates a uniform

distribution on (0,1); from this it is possible to generate random variates from other distributions.

Assuming that the system being modeled consists of multiple components with constant failure rates, and that the common cause event rates are also constant, then the total event rate can be written as sum of the individual event rates for all events:

$$\lambda_{total} = \sum_{i=1}^{n} \lambda_i$$

where the set of *n* possible events is exhaustive and mutually exclusive. In simulation, the time to the next event is generated by first selecting a random variate *z* from the uniform distribution U(0,1) and then performing the following transformation [Law and Kelton, 1991; Leemis and Park, 2005]:

$$t_{next} = t - \frac{1}{\lambda_{total}} \ln z$$

For example, " x = -log (rand (1,10))" in MATLAB produces a vector of ten realizations of an exponential distribution with parameter λ = 1. Application of this approach will be shown in the example problems in the following chapters. Given that an event occurred at time *t*, the specific event that occurred can be simulated using the relative probabilities:

$$P(Event\ i) = \frac{\lambda_i}{\lambda_{total}}$$

As an example of how to implement this approach in simulation, suppose that there are five possible events with the following rates and corresponding probabilities, given that an event has occurred:

Event #	Rate	Probability of occurrence, given that an event occurs
1	1 / hour	0.1
2	3 / hour	0.3
3	2 / hour	0.2
4	2 / hour	0.2
5	2 / hour	0.2
Sum	10 / hour	1.0

Table 4 – Event probabilities with constant failure rates

Also suppose in this context that each unique event can only change the system state once, representing the failure of one or more components in a non-repairable system; the components that fail in response to an event will remain failed for the duration of the simulation. Therefore, if the simulation generates a particular event more than once, it will be treated as a "null event." Starting at time zero, a random variate is drawn from the uniform distribution and the calculation shown above generates the time to the next event. Then another random variate is drawn from the uniform distribution to determine which event occurred, based on the relative probabilities of the events. The following table shows a typical sequence of times and events based on drawing the random variates shown:

Generating event times			Generating event types		
Uniform Random	Interarrival Time	Cumulative Time	Uniform Random	Event #	Remarks
0.010	0.459	0.459	0.011	1	
0.780	0.025	0.484	0.718	4	
0.220	0.152	0.636	0.917	5	
0.902	0.010	0.646	0.026	1	Null event
0.299	0.121	0.767	0.808	5	Null event
0.640	0.045	0.811	0.359	2	
0.602	0.051	0.862	0.126	2	Null event
0.250	0.139	1.001	0.616	4	Null event
0.235	0.145	1.146	0.322	2	Null event
0.069	0.267	1.412	0.293	2	Null event

Table 5 – Example result, constant event rates

Chapter 3: Discrete Event Simulation of Common Cause Failure – A Simple Example

3.1 Chapter Introduction:

This chapter will compare the traditional analytic approach for evaluating the impact of common cause failures with the discrete event simulation approach, using a simple example to illustrate the method. Following chapters will provide additional mathematical rigor to the simulation method and extend it to more complex examples.

Work done by Mosleh and others during the 1980s and 1990s developed best practices and procedures for incorporating common cause failures into probabilistic risk assessment (Mosleh, et. al., 1989; Mosleh, 1991). As noted in the previous chapter, the specific sequence of steps in the common cause failure analysis procedure vary somewhat by author; however, for purposes of the example, the steps are numbered as follows:

1	System familiarization and statement of assumptions.
2	Definition of "top events" and scenarios.
3	Development of logical models of the system.
4	Identification of common cause component groups.
5	Development of an algebraic model that determines the rate at which the "top event" occurs based on the rates of the basic events.
6	Estimation of basic event parameters.
7	Quantification of overall system reliability.

Table 6 – Process for incorporating common cause failure analysis in a probabilistic risk assessment

The primary difference between the proposed discrete event simulation approach and the traditional analytic approach is in step 5. Rather than developing an algebraic model, simulation is used to determine the probability of failure (or continued operation) as a function of time. Because discrete event simulation is done in the time domain, it is naturally well suited to incorporating time-varying event rates. Note that other than step 5, the analytic and simulation-based approaches are similar. In either approach, it is important to start with a correct description of the top event and potential basic events that cause the top event, with appropriate Boolean logic. Either approach also requires an estimate of the basic event rates, although the simulation-based approach is flexible in accommodating time-varying basic event rates. The remainder of this chapter provides a very simple example to illustrate the discrete event simulation method in comparison to the traditional analytic approach. Following chapters incorporate additional features in the simulation and apply the method to problems of increasing complexity.

3.2 Traditional Analytic Approach

Start by considering a simple system consisting of two components operating in parallel as shown:

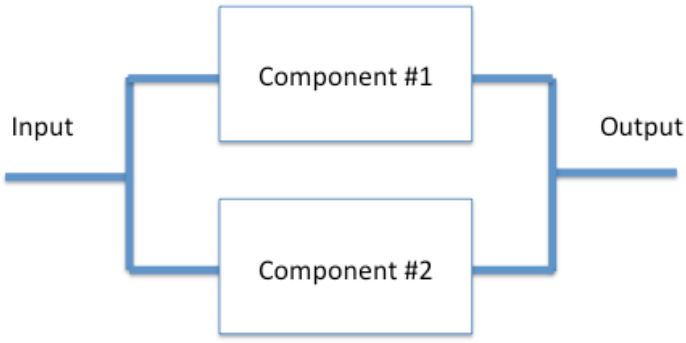


Figure 6 – A simple two-component parallel system

This two-component block is representative of many typical redundant systems embedded in a larger system; for example, two centrifugal pumps drawing from a common supply tank and feeding a downstream system. The notion of "redundancy" implies that either pump can provide the entire required capacity of the downstream system for the defined level of operability. This could either be for normal operation or for a minimum level of operation required for safety.

3.2.1. STEP 1 - System Familiarization and Statement of Assumptions:

For the purpose of this exercise, assume that the system includes only the two components shown, and that the components are identical, with identical failure rates. Assume that upstream and downstream components are either perfectly reliable or are not within the scope of the analysis. Assume further that each component can only be in two possible states: operable or not operable. Assume that the transition from operable to not operable occurs nearly instantaneously. Assume that the system is operable if either component is operable, and that the failure rate of each component is constant at rate λ (this assumption will be relaxed later). Assume also that there is a non-zero possibility that both components will fail simultaneously due to a common cause, and that the probability of common cause failure is modeled using the Beta factor model.

3.2.2 STEP 2 - Definition of top events and scenarios:

In this case, the "top event" to be modeled is that the system becomes non-operable due to failure of both components, and that this can occur in two ways:

a. Due to independent failures of both of the two components during the time period under examination.

b. Through common-cause simultaneous failure of both components during the time period under examination.

3.2.3. STEP 3 - Development of logical models of the system:

A common logical model used in probabilistic risk assessment is the fault tree. A fault tree model of the two-component parallel system is shown below:

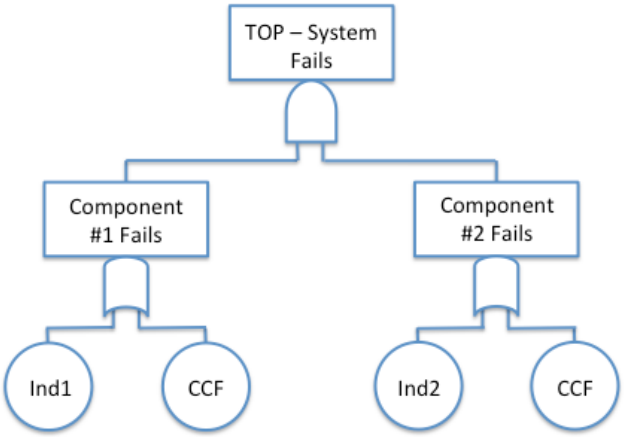


Figure 7 – Fault tree for simple two component parallel system

As suggested by Mosleh (1991), the fault tree is extended one level below the individual component failure level to allow identification of more than one potential cause of failure; the component failure is then considered an intermediate event that can be caused by one or more basic events. Conceptually the basic event level could include multiple potential root causes of failure, but in this case two basic events are allowed for each component: individual component failure of either component, and simultaneous failure of both components due to a common cause.

3.2.4. STEP 4 - Identification of Common Cause Component Groups:

In general, common cause groups consist of two or more components that could plausibly fail simultaneously due to a common root cause; this definition necessarily requires the application of judgment by the analyst. Two or more identical or similar

components arranged in parallel to serve a common function would typically form a common cause group. It would also be plausible to group components that share a common location or environment. In this simple case, component #1 and component #2 are deemed to form a common cause component group.

3.2.5. STEP 5 - Development of an algebraic model (analytic approach):

Applying the Beta Factor model developed in the previous chapter, the rate of failure for either component from the combination of individual and common causes is:

$$\lambda_1 = \lambda_2 = \lambda = \lambda_i + \lambda_c$$

$$\lambda_c = \beta\lambda; \quad \lambda_i = (1-\beta)\lambda$$

Failure of the system occurs when both components are failed.

3.2.6. STEP 6 - Estimation of basic event parameters:

For real systems, the estimation of basic event parameters can be the most difficult part of the analysis, since estimates are typically based on sparse data. For purposes of this example, assume the following values for each of the parameters:

$\lambda = 10^{-3}$ failures per hour (total failure rate for either individual component)

$\beta = 0.1$

$\lambda_i = 9 \times 10^{-4}$

$\lambda_c = 1 \times 10^{-4}$

3.2.7. STEP 7 - Quantification of overall system reliability:

Plugging the event parameters into the algebraic model developed in Chapter 2, overall system reliability can be expressed as the probability of system survival to time *t*:

$$R_{sys} = (e^{-\lambda ct})(2e^{-\lambda it} - e^{-2\lambda it})$$

A plot of R_{sys} as a function of time is provided later in this chapter in Figure 8 and 9.

3.3 Discrete Event Simulation Approach, Step by Step

As noted before, the simulation-based approach is identical to the analytical approach until step 5, development of the algebraic model. Instead of developing an algebraic model that provides an analytic solution, the discrete event simulation method proceeds as follows:

1 – 4, 6, 7	Steps 1-4 are identical for either method. As a practical matter, step 6 (quantification of basic event rates) is a necessary input to the simulation, and step 7 (quantification of overall system reliability) is an output of the simulation.
5(a)	Generate a system state vector that expresses component states as vector elements (1 for operable, 0 for inoperable).
5(b)	Input the individual component failure event rates and common cause event rates as elements of an event rate vector.
5(c)	Determine the total event rate, which is the sum of elements in the event rate vector.
5(d)	Determine the time to the next event by generating a random variate.
5(e)	Determine which specific event occurred at time t using the relative probabilities of all possible events evaluated at time t.
5(f)	Determine which component(s) failed. Adjust the system state vector, setting the failed component element values to zero.
5(g)	Determine whether the system is operable, using Boolean logic.
5(h)	If the system is still operable, make any needed adjustments to the event rate vector; for example, if the occurrence of a particular event changes the rates of other events.
5(i)	Generate time to the next event.
5(j)	Continue until the system is inoperable or until some fixed time.
5(k)	Repeat the simulation for a sufficient number of runs to generate meaningful statistics.

Table 7 – Discrete Simulation Steps

The remainder of this section provides an example of the application of these steps to the two component parallel system using the beta factor model. Segments of MATLAB code that implement each of the steps are shown in a different font. At the end, the complete MATLAB code is provided with annotations and a plot comparing the analytical solution to the simulation solution.

a. Develop system state vector: For the two component parallel system, the system state vector at t=0 with both components operable is $\mathbf{x} = [\ x_1\ \ x_2\ ] =$ [1 1]. In MATLAB , the system is initialized with the expression:

```
SSV = [ 1 1 ];  % Initialize system state vector
```

In much of the text, I have used **x** as the system state vector, following the convention of Ross (1993). However, in the MATLAB code, **x** is called "SSV;" in developing and debugging code, it is good practice to use variable names that are somewhat descriptive. At time t=0, the simulation starts with both components functioning. As the simulation progresses and components fail, any of the following conditions are possible:

x = [0 1] Component #1 failed

x = [1 0] Component #2 failed

x = [0 0] Component #1 and #2 failed

b. Input the individual component failure event rates and common cause event rates as elements of an event rate vector:

```
lambda = 0.001;
beta = 0.1;

% Individual Component Failures:
  A = (1-beta)*lambda; % Component #1 failure rate
  B = (1-beta)*lambda; % Component #2 failure rate
```

```
C = beta*lambda; % Common cause failure rate
%
Event_rate_vector = [A B C];
```

c. Determine the total event rate:

```
Total_event_rate = sum (Event_rate_vector);
```

Note that the total event rate in this example is $2\lambda_i + \lambda_c = 0.0019$

d. Determine the time to the next event: This is a core step of the discrete event simulation process; determining random time to the next event. This is done in MATLAB by an expression such as:

```
t= t - 1 /Max_event_rate * log(rand);
```

A more detailed discussion of how to specify the "Max_event_rate" will be provided in a subsequent chapter. This rate must be greater than or equal to the sum of all of the individual event rates evaluated at any time within the scenario being evaluated, denoted in the MATLAB code as "Total_event_rate." The rationale for making this rate greater than the sum of the individual event rates is to accommodate the situation where event rates are changing with time. In the present example, since the individual event rates are treated as constant, it is simple to determine a "Max_event_rate" that is greater than this sum. The "Max_event_rate" used in this particular simulation is 0.005, which is substantially higher than the "Total_event_rate" of 0.0019. Using a "Max_event_rate" higher than the "Total_event_rate" will not impact the accuracy of the simulation, but will slow down the simulation by generating many null events. In this simple example, simulation speed is not a significant concern.

e. Determine which specific event occurred at time t using the relative probabilities of all possible events evaluated at time t:

```
Event_prob = Event_rate_vector / Max_event_rate;
Event_type = sum(rand > cumsum(Event_prob))+1;
```

"Event_prob" is the vector of probabilities that any specific event other than the null event occurred at the time *t*. In this case it is:

```
Event_prob =    [ 0.1800    0.1800    0.0200 ]
```

f. Determine which component(s) failed. Adjust the system state vector, setting the failed component element values to zero. This step begins with a definition of the possible events and creation of an "Event_matrix." The possible events are:

```
% Define possible events and consequences
E1 = [ 0 1 ]; % Individual Failure of Component #1
E2 = [ 1 0 ]; % Individual Failure of Component #2
E3 = [ 0 0 ]; % Simultaneous failure due to common cause
E4 = [ 1 1 ]; % Null Event
%

Event_matrix = [E1; E2; E3; E4];
```

The Event_matrix is then:

Event_matrix =

0	1
1	0
0	0
1	1

The system state vector is adjusted at the occurrence of each event by an element-wise multiplication of the row of the event matrix corresponding with the event type by the previous state vector:

```
SSV = Event_matrix (Event_type, 1:2).*SSV;
```

So, for example, if the previous system state vector was [1 0], corresponding to component #1 functioning and component #2 failed, and the event type is a failure of component #1, then the new system state vector will be:

$$\mathbf{x} = [\ 0\ 1\ ]\ .*\ [\ 1\ 0] = [\ 0\ 0\ ]$$

Note that ".*" represents an element-by-element multiplication of a row of the Event_matrix by previous system state vector. This approach has some advantages. It ensures that components, once failed, remain failed. Also, it doesn't require an adjustment of the failure rate of previously failed components, since a second failure of the same component has the same effect on the system as a null event; for example, a second occurrence of the failure of component #2 would produce the result:

$$\mathbf{x} = [\ 1\ 0\ ]\ .*\ [\ 1\ 0\ ] = [\ 1\ 0\ ]$$

g. Determine whether the system is operable, using Boolean logic. In this case, the system is operable if either of the parallel components is still functioning. In MATLAB, this is determined by taking the sum of the elements of the system state vector **x**:

```
System_operable = sum(SSV);
```

h. If the system is still operable, make any needed adjustments to the event rate vector; for example, if the occurrence of a particular event changes the rates of other events. In this example, the event rates do not change during the simulation. An example in which they do change will be described in a later chapter.

i. Generate time to the next event. In MATLAB conditional repetition of a segment of code is easily accomplished using a "while" statement:

```
while System_operable > 0
t= t - 1 /Max_event_rate * log(rand);

...

end
```

j. Continue until the system is inoperable or until some fixed time. In this example, there is no fixed end time to the simulation, but one could be provided by adding an additional "while" statement like:

```
while time < 24
```

This would be useful for example if one is primarily interested in reliability of an emergency backup system during the first 24 hours of operation, assuming that the primary system can be returned to operation or a safe shut-down of a plant can be accomplished in that time.

k. Repeat the simulation for a sufficient number of runs to generate meaningful statistics. In MATLAB, a desired number of runs can be specified using the "for" statement:

```
for k = 1:n
 ...
end
```

In this example, *n* was set equal to 1000, providing 1000 runs of the simulation. This provides a good "eyeball" convergence on the analytic result. A more rigorous treatment of statistical convergence is provided in a subsequent chapter.

3.4 Discrete Event Simulation - Tying it all Together

Each of the steps in the DES process was described above. The MATLAB code shown below is an example of how the parts of the process come together to produce a useful result.

```
clear all
clc
% Define possible events and consequences
E1 = [ 0 1 ]; % Individual Failure of Component #1
E2 = [ 1 0 ]; % Individual Failure of Component #2
E3 = [ 0 0 ]; % Simultaneous failure due to common cause
E4 = [ 1 1 ]; % Null Event
%
Event_matrix = [E1; E2; E3; E4];
% Initialize the system:
n = 1000; timevector = zeros(1,n);
lambda = 0.001;
beta = 0.1;
Max_event_rate = 0.005;
% Start simulation for n number of runs:
    for k = 1:n
        SSV = [ 1 1 ];  % Initialize system state vector
        System_operable = 1;
        t = 0;
        while System_operable > 0
        t= t - 1 /Max_event_rate * log(rand);
        % Determine the event rates at time t
        % Individual Component Failures:
        A = (1-beta)*lambda; % Component #1 failure rate
        B = (1-beta)*lambda; % Component #2 failure rate
        C = beta*lambda; % Common cause failure rate
        %
        Event_rate_vector = [A B C];
        %
        Total_event_rate = sum (Event_rate_vector);
        %
        % Determine which event occurred:
        Event_prob = Event_rate_vector / Max_event_rate;
        Event_type = sum(rand > cumsum(Event_prob))+1;
        SSV = Event_matrix (Event_type, 1:2).*SSV;

        % Determine Operability:

        System_operable = sum(SSV);
        end
        timevector (k) = t;
    end
    totaltime =sum(timevector);
avetime = totaltime/n;
timesort = sort(timevector);
y_axis_counter = 1:n;
```

```
hold on
p=plot (timesort, (n - y_axis_counter)/n );
set(p,'Color','blue','LineWidth',1.05)
xlim ([0,3000])
ylim ([0,1])
xlabel 'time'
ylabel 'probability system is operable'

% Plot analytic answer
t=0;
lambda = 0.001;
beta = 0.1;
lambda_i = (1-beta)*lambda;
lambda_c = beta*lambda;
R = zeros(1,3000);
%
while t<3000
    t = t+1;
     First_term = exp(-1*lambda_c*t);
     Second_term = 2*exp(-1*lambda_i*t)-exp(-2*lambda_i*t);
    R(t) = First_term*Second_term;
end
p=plot (R);
set(p,'Color','red','LineWidth',1.05)
```

3.5 Comparison of Simulation Result to Analytic Result

The following charts, Figure 8 and 9 were generated by the MATLAB code shown above, with the simulation result shown in blue and the analytic result shown in red. It can be seen that the two results are in close agreement. Figure 8 compares analytic and simulation runs over a long period of time, 3000 hours. If the primary concern in this scenario is achieving very high reliability over a short time while other systems are restored, the simulation can be adjusted to focus on that issue. By increasing the number of simulation runs to 50,000 and limiting the simulation to 24 hours with an additional conditional statement, and then re-plotting to focus on only the first 24 hours, Figure 9 is obtained, showing that the system is approximately 99.7% reliable in a 24 hour period.

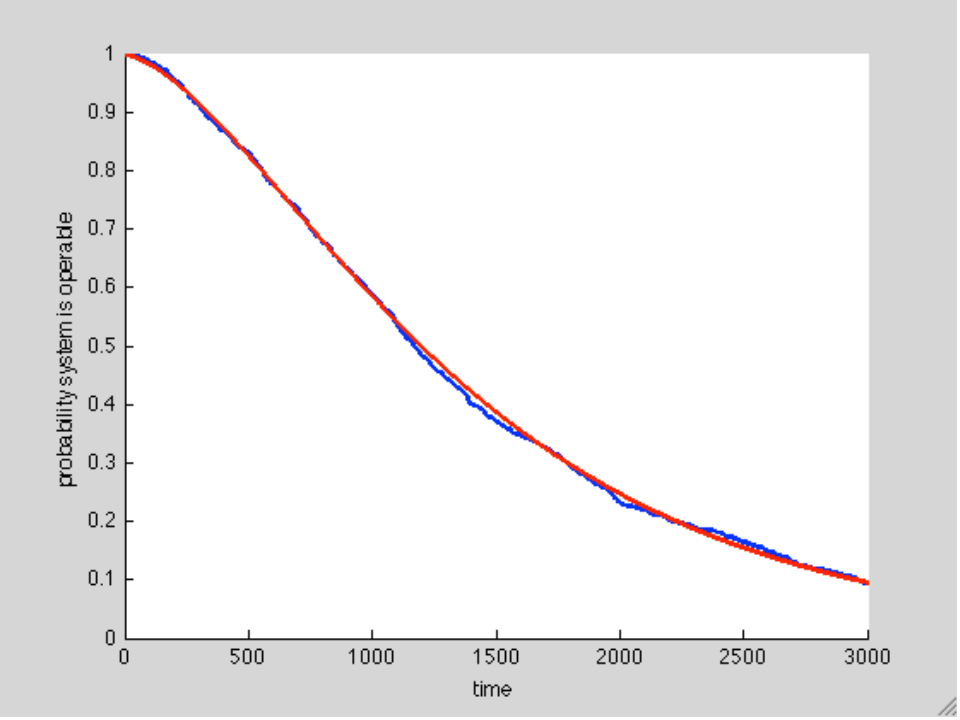


Figure 8 – Comparison of simulation result (blue) with analytic result (red) for a two-component parallel system with λ=0.001 and β = 0.1

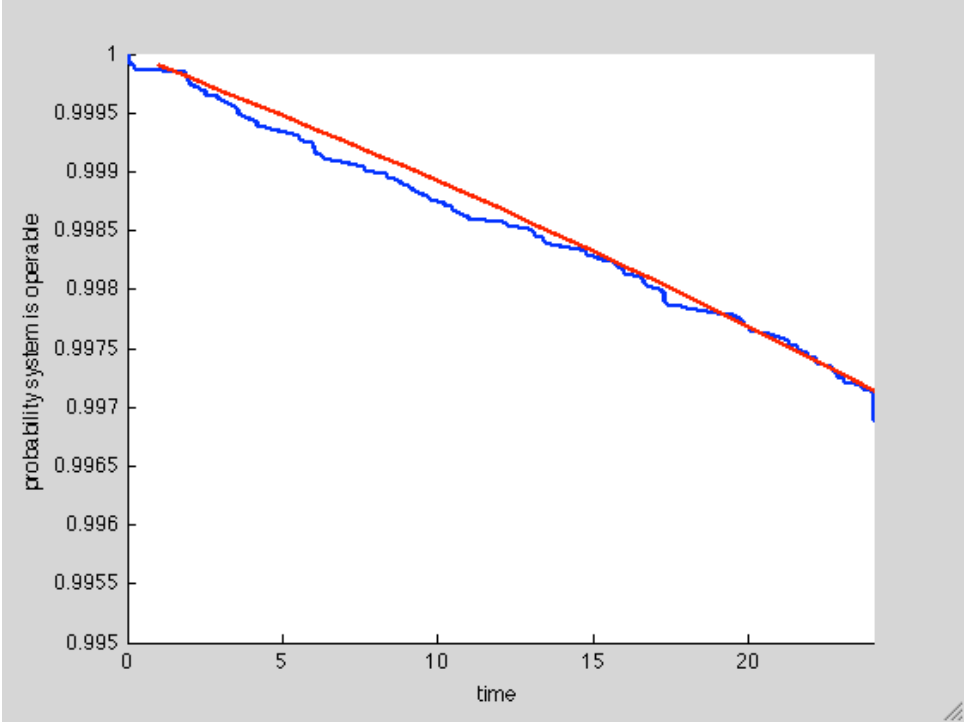


Figure 9 – Close-up comparison of simulation result (blue) with analytic result (red) for a two-component parallel system with λ=0.001 and β = 0.1. This is the upper left corner of Figure 8

This Chapter has given an overview of how to conduct a discrete event simulation of a simple system incorporating common cause failures. In this simple case, an analytic solution was readily obtainable, so a simulation isn't needed, although it is comforting to see that both methods produce compatible results. Subsequent chapters will extend the simulation approach to problems that aren't solvable analytically, showing the utility of the discrete event simulation method for solving hard problems in probabilistic risk assessment.

Chapter 4: Incorporating Failure on Demand

4.1 Chapter Introduction

As noted in section 2.5, both time-based and demand-based failures are of interest in analyzing common cause failures of a system, particularly in back-up or standby systems that are called upon to operate in the event of failure of a primary system. In addition to the consideration of failures that occur over time, there is a finite probability that one or more of the components of a system will fail on demand at time $t = 0$.

For this chapter, we will use as an example a system of four identical components, any two of which must be operable for the system to function. This system was chosen because there is a known analytic solution to which the simulation results can be compared, under the simplifying assumptions of constant component failure rate and component independence. The existence of an analytic solution under simplified conditions aids in validation of the model. The systems will be modeled using the Basic Parameter common cause failure model including both demand-based and time-based failure parameters, starting with a simple case of no dependent failures, and then extending to include common cause failures both at system start-up and during run time. In Chapter 5, this same example will be used, with a further extension to time-varying failure rates.

4.2 Four-Component System Analytic Solution with Independent Components

For a *k-out-of-n* system with identical and independent components, the probability of system survival to time t is a function of the individual component failures, including both probability of failure on demand and run-time failure. Denote the

probability that a particular component has survived to a particular time as $p(t)$; then the reliability of a *k-out-of-n* system is given as (Ross, 1993):

$$R(p) = \sum_{i=k}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

$$p = (1 - \lambda_{start}) e^{-\lambda_{run} t}$$

where λ_{start} is the probability of failure at start-up for an individual component and λ_{run} is the rate of failure of an individual component after time t=0, assuming an exponential distribution. Solving for the case that k = 2 and n = 4,

$$R(t) = 6((1 - \lambda_{start}) e^{-\lambda_{run} t})^2 (1 - (1 - \lambda_{start}) e^{-\lambda_{run} t})^2$$
$$+4((1 - \lambda_{start}) e^{-\lambda_{run} t})^3 (1 - (1 - \lambda_{start}) e^{-\lambda_{run} t}) + ((1 - \lambda_{start}) e^{-\lambda_{run} t})^4$$

This result will be used in the following section for comparison with simulation results. Note that there is also an analytic solution for the 2-out-of-4 case in which component failures are not independent, but it is much more complex, involving a polynomial with 64 terms. As will be shown, the simulation approach doesn't change significantly in complexity as dependent failures are introduced.

Under the simplifying assumption of component independence, the following basic parameters are obtained for probability of failure:

$\lambda_{s1} = (\lambda_{start}) * (1-\lambda_{start})^3$, failure of exactly one component at start-up
$\lambda_{s2} = (\lambda_{start})^2 * (1-\lambda_{start})^2$, failure of exactly two components at start-up
$\lambda_{s3} = (\lambda_{start})^3 * (1-\lambda_{start})$, failure of exactly three components at start-up
$\lambda_{s4} = (\lambda_{start})^4$, failure of all four components at start-up
$\lambda_{r1} = \lambda_{run}$, rate of single component failure during run-time
$\lambda_{r2} = 0$, rate of two-component simultaneous failure during run-time
$\lambda_{r3} = 0$, rate of three-component simultaneous failure during run-time
$\lambda_{r4} = 0$, rate of four-component simultaneous failure during run-time

In the case of common cause failures, the relationships shown above are more complex. The parameters λ_{r2}, λ_{r3}, and λ_{r4} are generally non-zero, and all of the parameters are

determined based on historical data for failures of similar components in similar systems, expert judgment in the typical case of sparse or questionably relevant data, and the specific common cause failure model being used.

4.3 Modeling and Simulation Approach

Start by defining the system state vector **x** as:

$$\mathbf{x} = [\, x_1 \,,\, x_2 \,,\, x_3 \,,\, x_4 \,]$$

where $x_i = 1$ for each functioning component and zero for each failed component. The structure function $\phi\,(\mathbf{x})$ is then defined as:

$$\phi\,(\mathbf{x}) = 1 \text{ if } x_1+x_2+x_3+x_4 \geq 2\text{; zero otherwise}$$

In other words, if two or more components are functional, then the system is functional. To implement this model in MATLAB, start by defining an event matrix **E** consisting of all the possible events that can occur to the system. An event is defined as the failure of one or more components, plus the "null event" in which no components fail. The events are chosen and defined in such a way as to be both exhaustive and mutually exclusive. Based on this definition, there are sixteen possible events that can occur either at start-up or during operation: four possible single component failures; six possible two-component failures; four possible three-component failures; a four-component failure; and the null event. Each possible event is a row in the event vector:

E =

0	1	1	1	failure of component #1
1	0	1	1	failure of component #2
1	1	0	1	failure of component #3
1	1	1	0	failure of component #4
0	0	1	1	failure of component #1 and #2
0	1	0	1	failure of component #1 and #3
0	1	1	0	failure of component #1 and #4

1	0	0	1	failure of component #2 and #3
1	0	1	0	failure of component #2 and #4
1	1	0	0	failure of component #3 and #4
0	0	0	1	failure of component #1, #2, and #3
0	0	1	0	failure of component #1, #2, and #4
0	1	0	0	failure of component #1, #3, and #4
1	0	0	0	failure of component #2, #3, and #4
0	0	0	0	failure of all four components
1	1	1	1	null event

At the time of initial demand, t = 0, there is a finite vector **P** that defines the probabilities that one of the sixteen possible events will occur. This collection of events defines all of the possible ways that the system can behave at start up. Since the events, including the null event, are defined as exhaustive and mutually exclusive, the sum of the elements of **P** is unity:

$$\sum_{i=1}^{n} P_i = 1$$

where each term provides the probability of occurrence of the event in the corresponding row of the event matrix **E**. The probability of the null event P_{null} is defined as:

$$P_{null} = 1 - \sum_{i=1}^{n-1} P_i$$

The actual values of the elements of **P** are estimated based on relevant failure data from similar systems, expert opinion, and assumptions embedded in a common cause failure model. In the simulation, a failure-on-demand event is generated at time t=0, and run-time failures are generated at times t>0 according to the probabilities given in the probability vector **P**. The event may be the failure of one or more components, the event that one or more already-failed components remain in the failed condition, or the null

event. The next two sections describe the simulation process in greater detail for both the failure-on-demand and run-time failure cases.

4.4 Simulation of Failure on Demand

Suppose that each component has an independent failure-on-demand probability λ_{start}=10%, and there are no common-cause failures. In this case, the vector **P** is:

```
P =
Columns 1 through 8
0.0729    0.0729    0.0729    0.0729    0.0081    0.0081    0.0081    0.0081

Columns 9 through 16
0.0081    0.0081    0.0009    0.0009    0.0009    0.0009    0.0001    0.6561
```

where columns 1 through 4 represent failure of exactly one component, columns 5 through 10 represent the six possible combinations of failure of exactly two components, columns 11 through 14 represent failure of exactly three components, column 15 represents failure of all four components, and column 16 is the null event.

To simulate failure on demand, it is necessary to generate a random variate to determine on each simulation run which of the possible start-up events occurred. This is done using the following MATLAB code:

```
ET = sum (rand>cumsum(P))+1
```

where "ET" is the event type corresponding to the indicator *i* in the vector **P**, and to the corresponding row of the event matrix **E**. To explain this in greater detail, "cumsum (**P**)" is a vector that is the cumulative sum of the elements of **P**. The "cumsum" of **P** is then given by:

```
Columns 1 through 8
0.0729    0.1458    0.2187    0.2916    0.2997    0.3078    0.3159    0.3240
```

```
Columns 9 through 16
0.3321    0.3402    0.3411    0.3420    0.3429    0.3438    0.3439    1.0000
```

The effect of this line of code is to pick one of the possible events at random based on the probability of occurrence of that event. For example, in the vector cumsum(**P**), if "rand" = 0.3221, then event type, ET = 8, corresponding to the 8th row of the event matrix. Next, generate an occurrence vector, **OV** of zeros equal in length to the possible number of events, and set the *i*th element equal to one. Multiply this vector by the event matrix:

```
OV = zeros(1,16); % Initialize Occurrence vector
OV (ET) = 1;
EC = OV*E;
```

Note that this is a matrix multiplication, where the dimensions of the occurrence vector and the event matrix must agree. **OV** is of dimension 1x16, and **E** is of dimension 16x4, with rows representing the 16 possible events and columns representing the four individual components. The event consequence "**EC**" is then a 1x4 row vector picked from the event matrix representing the components that either fail or remain in a failed state based on the occurrence of that event.

The next step in the simulation is to multiply the event consequence vector **EC** element-wise by the system state vector **x**. Element-wise multiplication is occasionally known as the Hadamard product (Styan, 1973), and is represented as ".*" Assume that at some time prior to the start of the simulation, the system state vector is:

$$\mathbf{x} = [\ 1\ ,\ 1\ ,\ 1\ ,\ 1\ ]$$

In other words, all components are believed to be operational at some time prior to demand. Suppose that the random number generator "rand" produces 0.3221; the corresponding event type is E8, and the event consequence is:

$$\mathbf{EC} = [\ 1\ ,\ 0\ ,\ 0\ ,\ 1\ ]$$

This is the failure on demand of components #2 and #3. The element-wise product of **x** and **EC** is:

$$\mathbf{x}_{post} = \mathbf{x}_{prior} .* \mathbf{EC} = [\ 1\ ,\ 0\ ,\ 0\ ,\ 1\ ]$$

In this case, the system is still operable with two components, and therefore the structure function, $\phi\ (\mathbf{x}) = 1$. The simulation code will check for system operability and will either continue to the next time step if the system is operable or end execution and record the time of failure for that run. Note that this method works for systems that do not have the possibility of repair or restoration. Once a component state is flipped from 1 to 0 it will remain in that state through all subsequent events, since 0*1 = 0.

4.5 Simulation of Run Time Failure

If the system is still operable following time t = 0, the simulation must generate the time to the next event. Suppose that we have the simple case of no common cause failures, and that individual components fail according to an exponential distribution with a failure rate of $\lambda_{r1} = 0.01$ per hour. The corresponding model parameters and event rates are:

```
%Basic parameters
lambda_r1 = 0.01;
lambda_r2 = 0;
lambda_r3 = 0;
lambda_r4 = 0;

% Event rates
R = zeros(1,16);
R(1) = lambda_r1;
R(2) = lambda_r1;
R(3) = lambda_r1;
R(4) = lambda_r1;
```

Since all of the events occur at a constant rate, the maximum event rate is constant and is the sum of the elements of **R**:

```
MER = sum(R); % Define the maximum total event rate, which is the
              % sum of all individual event rates evaluated over
              % the relevant time interval
```

Time to the next event is generated using:

```
t = t - 1 / MER * log(rand);
```

At time t, an event is generated using the same approach as in the failure on demand case, where:

```
R =
Columns 1 through 8
0.0100    0.0100    0.0100    0.0100         0         0         0         0

Columns 9 through 16
     0         0         0         0         0         0         0
```

The following code generates an event type based on a random variate:

```
EP = R / MER; % Determine probability of each event
ET = sum(rand>cumsum(EP))+1; % Event type
```

Suppose that the event type is ET = 8, the same event that occurred at start up. In this case, the event is equivalent to the null event, since:

$$\mathbf{x}_{post} = \mathbf{x}_{prior} .* \mathbf{EC} = [\ 1\ ,\ 0\ ,\ 0\ ,\ 1\ ] .* [\ 1\ ,\ 0\ ,\ 0\ ,\ 1\ ] = [\ 1\ ,\ 0\ ,\ 0\ ,\ 1\ ]$$

This is convenient, since it eliminates the need to adjust the probabilities of occurrence for events that have already occurred.

Following each time step, an event is generated, the system state vector **x** is re-computed using the technique above, the structure function ϕ (**x**) is evaluated, and if ϕ (**x**) = 1, the simulation continues to the next time step; otherwise the simulation run is ended and time to failure is recorded. A large number of runs are generated to provide statistics on reliability as a function of time.

4.6 Complete Simulation of a Four-Component System:

To pull together the discussion from the previous sections, this section describes the implementation a Four-Component system with the possibility of common cause failure. The underlying model for failure-on-demand in this example is the Beta factor model. The rationale for using this model is that each component has some probability of individual failure at start-up, and additionally there is a probability that all components may fail at start-up due to problem with the control system, an incorrect plant line-up, or a maintenance error common to all of the components. The underlying model used for run-time failure is the Binomial Failure Rate model, which assumes that failure of any single component can be caused in one of three ways: due to an independent failure, due to a non-lethal shock which causes failure with probability P, and due to lethal shock. As an intermediate step, the Beta and Binomial Failure Rate parameters are used to calculate the Basic Parameters, since these are a direct input to the probability vector **P**. Since the basic parameter model is the most general model, extension to other underlying models is fairly straightforward.

The complete MATLAB code to implement the simulation of this system is as follows:

```
% SIMULATION OF A 2-OUT-OF-4 SYSTEM

clear all % clears MATLAB from prior runs
clc % clears command window

% DEFINE ALL POSSIBLE EVENTS

% Single component failures
E1 = [ 0 1 1 1 ]; % Failure of component #1
E2 = [ 1 0 1 1 ]; % #2
E3 = [ 1 1 0 1 ]; % #3
E4 = [ 1 1 1 0 ]; % #4
% Two-component failures
```

```
E5 = [ 0 0 1 1 ]; % Failure of #1 and #2
E6 = [ 0 1 0 1 ]; % #1 and #3
E7 = [ 0 1 1 0 ]; % #1 and #4
E8 = [ 1 0 0 1 ]; % #2 and #3
E9 = [ 1 0 1 0 ]; % #2 and #4
E10= [ 1 1 0 0 ]; % #3 and #4
% Three-component failures
E11 = [ 0 0 0 1 ]; % Failure of #1, #2, and #3
E12 = [ 0 0 1 0 ]; % #1, 2, and 4
E13 = [ 0 1 0 0 ]; % #1, 3, and 4
E14 = [ 1 0 0 0 ]; % #2, 3, and 4
% Four-component failures
E15 = [ 0 0 0 0 ]; % Failure of all four components
% Null event
E16 = [ 1 1 1 1 ];

E = [E1;E2;E3;E4;E5;E6;E7;E8;E9;E10;E11;E12;E13;E14;E15;E16];
% E is the event matrix; identifies which components fail for
% each possible specific event

% FAILURE ON DEMAND: DETERMINE THE PROBABILITIES FOR EACH EVENT

% Assume the beta factor model is the underlying model
lambda_s_ind = 0.1;  % probability of independent failure at start
lambda_s_c = 0.0;  % probability of CCF at start

lambda_s1 = (1-lambda_s_c)*lambda_s_ind*(1-lambda_s_ind)^3;
lambda_s2 = (1-lambda_s_c)*lambda_s_ind^2*(1-lambda_s_ind)^2;
lambda_s3 = (1-lambda_s_c)*lambda_s_ind^3*(1-lambda_s_ind);
lambda_s4 = lambda_s_c+(1-lambda_s_c)*lambda_s_ind^4;

% Event probabilities
P = zeros (1,16);
P(1) = lambda_s1; P(2) = lambda_s1; P(3) = lambda_s1;
P(4) = lambda_s1;
%
P(5) = lambda_s2; P(6) = lambda_s2; P(7) = lambda_s2;
P(8) = lambda_s2; P(9) = lambda_s2; P(10)= lambda_s2;
%
P(11) = lambda_s3; P(12) = lambda_s3; P(13) = lambda_s3;
P(14) = lambda_s3;
P(15) = lambda_s4;
P(16) = 1 - sum(P);

% FAILURE DURING RUN: DETERMINE THE RATES FOR EACH EVENT

% assume the binomial failure rate is the underlying model
lambda_r_ind = 0.01; % rate of independent failure
mu = 0.0; % rate of non-lethal shocks
Prob = 0.0; % prob of comp failure given non-lethal shock
omega = 0.0; % rate of lethal shocks

%Basic parameters
lambda_r1 = lambda_r_ind + mu*Prob*(1-Prob)^3; % rate of single failure
lambda_r2 = mu*Prob^2*(1-Prob)^2; % rate of double failure
lambda_r3 = mu*Prob^3*(1-Prob);
lambda_r4 = mu*Prob^4 + omega;
```

```
% Event rates
R = zeros(1,16);
R(1) = lambda_r1; R(2) = lambda_r1; R(3) = lambda_r1;
R(4) = lambda_r1;
%
R(5) = lambda_r2; R(6) = lambda_r2; R(7) = lambda_r2;
R(8) = lambda_r2; R(9) = lambda_r2; R(10) = lambda_r2;
%
R(11) = lambda_r3; R(12) = lambda_r3; R(13) = lambda_r3;
R(14) = lambda_r3;
%
R(15) = lambda_r4;

MER = sum(R); % Define the maximum total event rate, which is the
              % sum of all individual event rates evaluated over
              % the relevant time interval


% INPUT THE NUMBER OF SIMULATION RUNS

n= 10000; % number of runs
timevector = zeros(1,n); % A vector which collects the failure
                         % time in each run


% START THE SIMULATION
for k =1:n
    SSV = ones (1,4); % Initialize system state vector
    t = 0; %Reset system clock
    % possible failure on demand:
    OV1 = zeros(1,16); % Initialize Occurrence vector
    z1 = rand;
    ET1 = sum(z1>cumsum(P))+1;% Event type
    OV1 (ET1) = 1;
    EC1 = OV1*E; % Event consequence - selects a row from E
                 % corresponding to the event type
    SSV = SSV .* EC1;

    % failure time during run:
    while sum(SSV)>1
        OV2 = zeros(1,16); % Initialize Occurrence vector
        t = t - 1 / MER * log(rand); % Generate time to next event
        EP = R / MER; % Determine probability of each event
        z2 = rand;
        ET2 = sum(z2>cumsum(EP))+1; % Event type
        OV2 (ET2) = 1; % Puts a one in the element of OV corresponding
                       % to the event type
        EC2 = OV2*E; % Event consequence - selects a row from E
                     % corresponding to the event type
        SSV = SSV .* EC2;
    end
    timevector(k) =t; % collects time to failure from each run
    tmax = max(timevector);
end

% PLOT THE SIMULATION RESULTS
```

```
timesort = sort(timevector); % puts the failure times in
                             % increasing order
y_axis_counter = 1:n;

hold on
p = plot (timesort, (n-y_axis_counter)/n);
set(p,'Color','blue','LineWidth',1.05)

% define x and y axes; comment out the undesired x and y limits

%xlim ([0,tmax])
%ylim ([0,1])
xlim ([0,75])
ylim ([.8,1])
xlabel 'time'
ylabel 'probability system is operable'


% PLOT THE ANALYTIC RESULT TO COMPARE AGAINST SIMULATION
g = int16(max(timesort));
AR = ones(1,g);
t=1;

for k=1:g
A = (1-lambda_s_ind)*exp(-lambda_r1*t);
AR(k)= 6*A^2*(1-A)^2 + 4*A^3*(1-A) + A^4;
t = t+1;
end

p=plot (AR);
set(p,'Color','red','LineWidth',1.05)
```

4.7 Simulation Output and Comparison to Analytic Results:

This code generates the following plots, which show agreement between the simulation and the analytic result for the simplest case of no common-cause failures, lending confidence in the validity of the model.

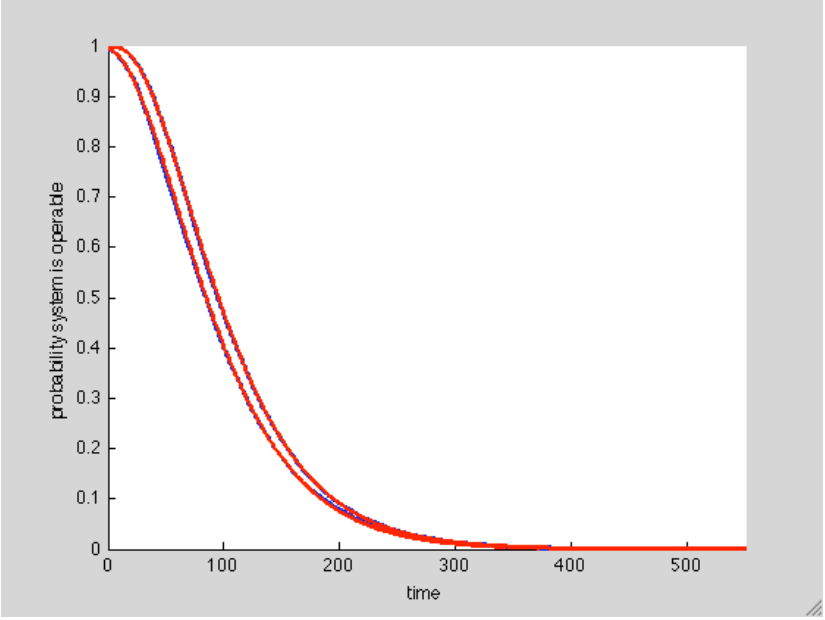


Figure 10 – Comparison of Simulation Result (blue) with Analytic Result (red) for a 2-out-of-4 system. Upper curves have $\lambda_{s_ind} = 0$; lower curves have $\lambda_{s_ind} = 0.1$

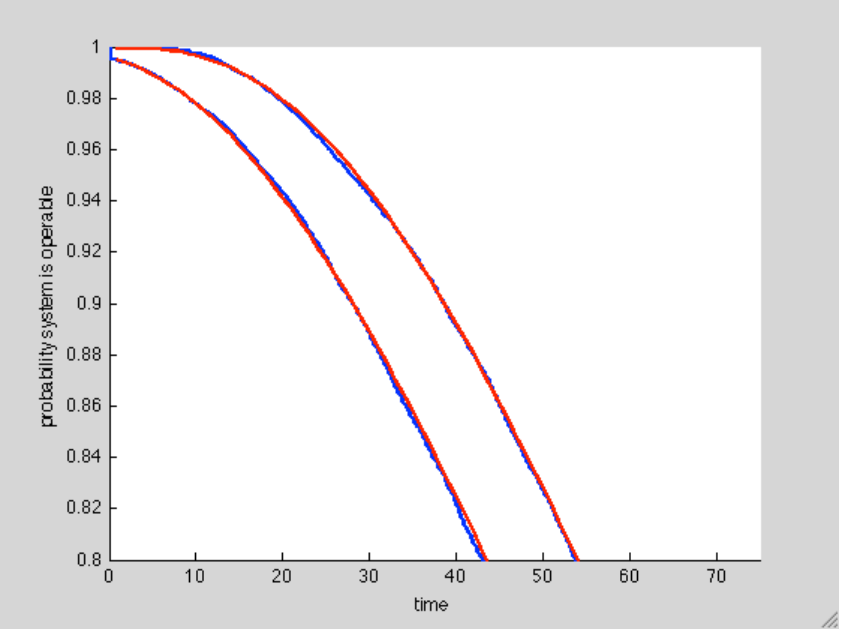


Figure 11 – Close-up Comparison of Simulation Result (blue) with Analytic Result (red) for a 2-out-of-4 system. Upper curves have $\lambda_{s_ind} = 0$; lower curves have $\lambda_{s_ind} = 0.1$

To show the impact of Common Cause Failures, the following figures show a comparison of the analytic and simulation results above to the following alternative scenario:

Failure on demand:

$\lambda_I = 0.1$; $\beta = 0.1$

Run time failure:

$\lambda i = 0.01$; $\mu = 0.005$ $P = 0.3$; $\omega = 0.0005$

```
% Assume the beta factor model is the underlying model
lambda_s_ind = 0.1;  % probability of independent failure at
start
lambda_s_c = 0.1;  % probability of CCF at start

% assume the binomial failure rate is the unerlying model
lambda_r_ind = 0.01; % rate of indepenent failure
mu = 0.005; % rate of non-lethal shocks
Prob = 0.3; % prob of comp failure given non-lethal shock
omega = 0.0005; % rate of lethal shocks
```

This results in the following probability vectors for start-up and run-time failures:

P =
Columns 1 through 8
0.0656 0.0656 0.0656 0.0656 0.0073 0.0073 0.0073 0.0073
Columns 9 through 16
0.0073 0.0073 0.0008 0.0008 0.0008 0.0008 0.1001 0.5905

R =
Columns 1 through 8
0.0105 0.0105 0.0105 0.0105 0.0002 0.0002 0.0002 0.0002
Columns 9 through 16
0.0002 0.0002 0.0001 0.0001 0.0001 0.0001 0.0005 0

Results of the modeling using these parameters are shown in Figure 11 below, in comparison to the analytical result with only independent run-time failures.

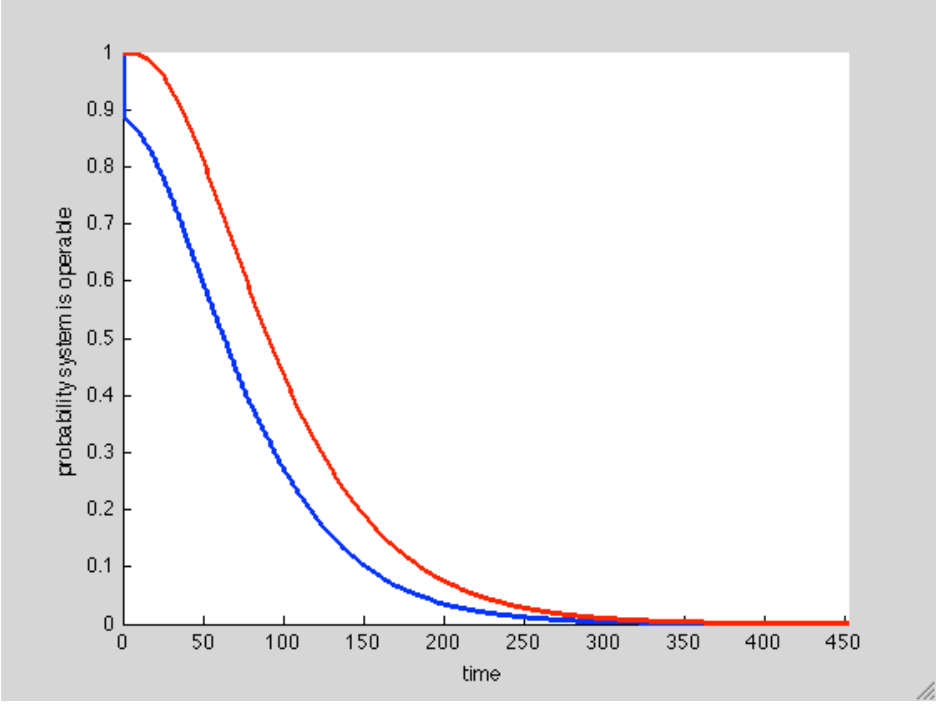


Figure 12 – Comparison of Simulation Result (blue) incorporating both start-up and run-time common cause failures to Analytic Result (red) incorporating only independent run-time failures for a 2-out-of-4 system. The drop at t=0 for the blue curve represents probability of failure on demand.

4.8 Summary

The simulation model as constructed has the flexibility to accommodate a range of assumptions about the common cause failure parameters lambda, since the code enables the user to assign any desired values to λ_{s1}, λ_{s2}, λ_{s3}, λ_{s4}, λ_{r1}, λ_{r2}, λ_{r3}, and λ_{r4}. The modeling approach accommodates time-varying event rates, as will be shown in the next chapter, and is reasonably extensible to systems with larger number of components by expanding the dimensions of the event matrix E; this will be shown in Chapter 6, with an example that includes eleven components in three common cause failure groups.

Chapter 5: Incorporating Time-Varying Event Rates

5.1 Chapter Introduction

As noted in Chapter 1, the models of common cause failure found in textbooks and handbooks incorporate the exponential failure model; in other words, the basic events that might lead to system failure occur at time-independent rates (Hockstad and Rausand, 2008). This assumption is a mathematical convenience that enables an analytic solution to many problems, although we have seen in some of the more complex examples that even with this assumption a closed-form solution can be difficult to calculate due to the multiplicity of terms in the equation. The ability to accommodate time-varying failure rates is one of the key advantages of using the discrete event simulation method. This chapter will provide the theoretical basis for incorporating time varying failure rates, and will provide an example of how to incorporate time-varying failure rates in a MATLAB-based simulation.

5.2 Thinning approach to incorporating time varying event rates

Consider a system that includes components with time varying failure rates, such that the total event rate in the simulation is a function of time. This system evolves according to a non-homogenous Poisson process, where the total event rate is expressed as $\lambda_{total}(t)$. In our previous simulation examples, we generated random events based on a constant maximum event rate; with time varying event rates a different approach is needed. Fortunately, there is a relatively simple approach using the "thinning method" (Law and Kelton, 1991; Lewis and Shedler, 1979). In this approach, the maximum event

rate which we will call λ_{max} is set to a constant level that exceeds the actual event rate over the whole period of the simulation, $0 \leq t \leq T$, and then individual events generated by an exponential random variate are accepted or rejected based on the ratio of the instantaneous event rate evaluated at time *t* to the maximum event rate. A rejected event is equivalent to the "null event" described earlier. Mathematically this can be described as follows (Ross, 1993):

$$\lambda_{total}(t) \leq \lambda_{max} \qquad \text{for all } t \leq T$$

If an event is generated by the simulation at time t by an exponential random variate with constant rate λ_{max}, then the relative probability of the null event is expressed by:

$$P_{null}(t) / \sum_{i=1}^{n} P_i(t) = 1 - \lambda_{total}(t) / \lambda_{max}$$

where P_i are the elements of the probability vector **P** denoting the probabilities of all possible *n* events including the null event, evaluated at time t. The thinning approach is described pictorially in Figure 13

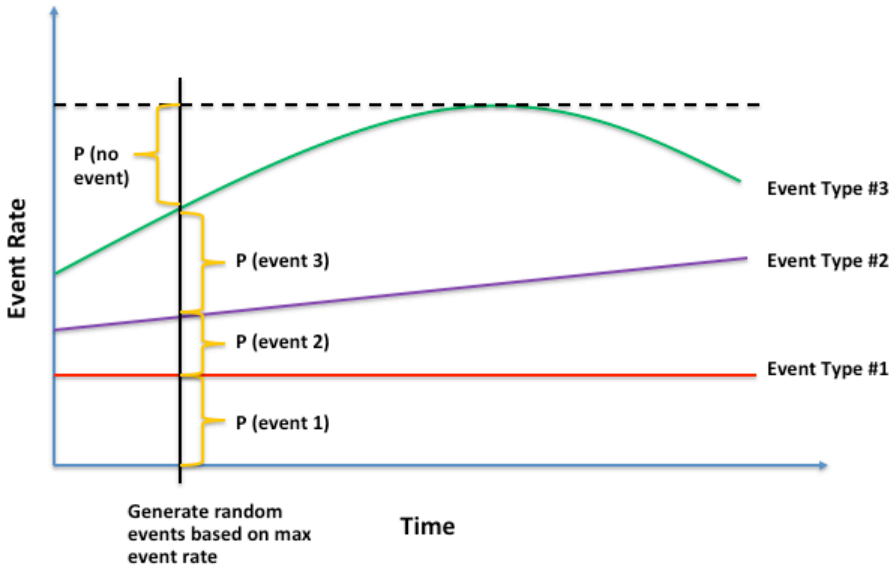


Figure 13 – Graphical representation of thinning technique

5.3 An Example of a k-out-of-n system with Weibull Failure Rate

Suppose that the four-component system introduced in Chapter 4 has independent component failures during run time that can be described by the Weibull distribution, such that the failure rate λ_{r1} is a function of t:

$$\lambda_{r1}(t) = \alpha {\lambda_w}^{\alpha} t^{\alpha-1}$$

where λ_w and α are parameters of the Weibull distribution. For $\alpha > 1$, this represents an increasing failure rate over time, corresponding to an aging process. Incorporation into the MATLAB simulation code developed in Chapter 5 is fairly straightforward. First, it is necessary to determine an appropriate maximum event rate, λ_{max}. In the case of a steadily increasing failure rate, this can be done by evaluating λ_{total} at the maximum time T that is of interest in the simulation. If the variation of λ_{total} over time is more complex or includes discontinuities, it may be necessary to evaluate λ_{total} more carefully as a function of time to ensure that a large enough value of λ_{max} is chosen. Choosing a too-small value of λ_{max} will lead to erroneous simulation results, while choosing a value that is larger than necessary will increase the run time of the model; with fast computers run time may not be a significant concern. Note that since the model now includes time-dependent event rates, the vector **R** of all event rates must be evaluated as a function of *t* at the time of each event. Therefore, the lines of code that calculate **R** must be inserted inside the "while" loop that generates run-time events. Otherwise, the code is similar to what was developed in Chapter 5. Picking some specific values, suppose that $\alpha = 1.5$ and $\lambda_w = 0.01$. The modified MATLAB code for this example is as follows:

```
% SIMULATION OF A 2-OUT-OF-4 SYSTEM

clear all % clears MATLAB from prior runs
clc % clears command window

% DEFINE ALL POSSIBLE EVENTS

% Single component failures
E1 = [ 0 1 1 1 ]; % Failure of component #1
E2 = [ 1 0 1 1 ]; % #2
E3 = [ 1 1 0 1 ]; % #3
E4 = [ 1 1 1 0 ]; % #4
% Two-component failures
E5 = [ 0 0 1 1 ]; % Failure of #1 and #2
E6 = [ 0 1 0 1 ]; % #1 and #3
E7 = [ 0 1 1 0 ]; % #1 and #4
E8 = [ 1 0 0 1 ]; % #2 and #3
E9 = [ 1 0 1 0 ]; % #2 and #4
E10= [ 1 1 0 0 ]; % #3 and #4
% Three-component failures
E11 = [ 0 0 0 1 ]; % Failure of #1, #2, and #3
E12 = [ 0 0 1 0 ]; % #1, 2, and 4
E13 = [ 0 1 0 0 ]; % #1, 3, and 4
E14 = [ 1 0 0 0 ]; % #2, 3, and 4
% Four-component failures
E15 = [ 0 0 0 0 ]; % Failure of all four components
% Null event
E16 = [ 1 1 1 1 ];

E = [E1;E2;E3;E4;E5;E6;E7;E8;E9;E10;E11;E12;E13;E14;E15;E16];
% E is the event matrix; identifies which components fail for
% each possible specific event

% FAILURE ON DEMAND: DETERMINE THE PROBABILITIES FOR EACH EVENT

% Assume the beta factor model is the underlying model
lambda_s_ind = 0;  % probability of independent failure at start
lambda_s_c = 0;  % probability of CCF at start

lambda_s1 = (1-lambda_s_c)*lambda_s_ind*(1-lambda_s_ind)^3;
lambda_s2 = (1-lambda_s_c)*lambda_s_ind^2*(1-lambda_s_ind)^2;
lambda_s3 = (1-lambda_s_c)*lambda_s_ind^3*(1-lambda_s_ind);
lambda_s4 = lambda_s_c+(1-lambda_s_c)*lambda_s_ind^4;

% Event probabilities
P = zeros (1,16);
P(1) = lambda_s1; P(2) = lambda_s1; P(3) = lambda_s1;
P(4) = lambda_s1;
%
P(5) = lambda_s2; P(6) = lambda_s2; P(7) = lambda_s2;
P(8) = lambda_s2; P(9) = lambda_s2; P(10)= lambda_s2;
%
P(11) = lambda_s3; P(12) = lambda_s3; P(13) = lambda_s3;
P(14) = lambda_s3;
P(15) = lambda_s4;
P(16) = 1 - sum(P);
```

```
MER = 0.2;     % Define the maximum total event rate, which is the
               % sum of all individual event rates evaluated over
               % the relevant time interval


% INPUT THE NUMBER OF SIMULATION RUNS

n= 10000; % number of runs
timevector = zeros(1,n); % A vector which collects the failure
                         % time in each run


% START THE SIMULATION
for k =1:n
    SSV = ones (1,4); % Initialize system state vector
    t = 0; %Reset system clock
    % possible failure on demand:
    OV1 = zeros(1,16); % Initialize Occurrence vector
    z1 = rand;
    ET1 = sum(z1>cumsum(P))+1;% Event type
    OV1 (ET1) = 1;
    EC1 = OV1*E; % Event consequence - selects a row from E
                 % corresponding to the event type
    SSV = SSV .* EC1;

    % failure time during run:
    while sum(SSV)>1
        OV2 = zeros(1,16); % Initialize Occurrence vector
        t = t - 1 / MER * log(rand); % Generate time to next event

        % FAILURE DURING RUN: DETERMINE THE RATES FOR EACH EVENT

        % Find component independent failure rate as a fn of t
        % using Weibull distribution
        lambda_w = 0.01;
        alpha = 1.5;
        lambda_r_ind = alpha*(lambda_w^alpha)*(t^(alpha-1));

        % assume the binomial failure rate is the underlying model
        mu = 0.0; % rate of non-lethal shocks
        Prob = 0.3; % prob of comp failure given non-lethal shock
        omega = 0.0; % rate of lethal shocks

        %Basic parameters
        lambda_r1 = lambda_r_ind + mu*Prob*(1-Prob)^3; % rate of single
        %failure
        lambda_r2 = mu*Prob^2*(1-Prob)^2; % rate of double failure
        lambda_r3 = mu*Prob^3*(1-Prob);
        lambda_r4 = mu*Prob^4 + omega;

        % Event rates
        R = zeros(1,16);
        R(1) = lambda_r1; R(2) = lambda_r1; R(3) = lambda_r1;
        R(4) = lambda_r1;
        %
        R(5) = lambda_r2; R(6) = lambda_r2; R(7) = lambda_r2;
```

```
        R(8) = lambda_r2; R(9) = lambda_r2; R(10) = lambda_r2;
        %
        R(11) = lambda_r3; R(12) = lambda_r3; R(13) = lambda_r3;
        R(14) = lambda_r3;
        %
        R(15) = lambda_r4;
        R(16) = MER-sum(R);
        if R(16)<0 % This provides a warning if the max event rate
                   % is too small
            fprintf 'increase MER'
            pause
        end

        EP2 = R / MER; % Determine probability of each event
        z2 = rand;
        ET2 = sum(z2>cumsum(EP2))+1; % Event type
        OV2 (ET2) = 1; % Puts a one in the element of OV corresponding
                       % to the event type
        EC2 = OV2*E; % Event consequence - selects a row from E
                     % corresponding to the event type
        SSV = SSV .* EC2;
    end
    timevector(k) =t; % collects time to failure from each run
    tmax = max(timevector);
end

% PLOT THE SIMULATION RESULTS

timesort = sort(timevector); % puts the failure times in
                             % increasing order
y_axis_counter = 1:n;

hold on
p = plot (timesort, (n-y_axis_counter)/n);
set(p,'Color','blue','LineWidth',1.05)

% define x and y axes; comment out the undesiredx and y limits

xlim ([0,tmax])
ylim ([0,1])
%xlim ([0,75])
%ylim ([.8,1])
xlabel 'time'
ylabel 'probability system is operable'


% PLOT THE ANALYTIC RESULT TO COMPARE AGAINST SIMULATION
g = int16(max(timesort));
AR = ones(1,g);
t=1;

lambda_s_ind=0;
for k=1:g
A = (1-lambda_s_ind)*exp(-lambda_w*t);
AR(k)= 6*A^2*(1-A)^2 + 4*A^3*(1-A) + A^4;
t = t+1;
end
```

```
p=plot (AR);
set(p,'Color','red','LineWidth',1.05)
```

Figure 14 shows the system reliability as a function of time. The blue line is a simulation of a Weibull distribution of component failure rates, with $\alpha = 1.5$ and $\lambda_w = 0.01$. The red line is an analytic result assuming exponential failure rates, with $\lambda = 0.01$. Note that the Weibull distribution with $\alpha = 1.0$ and $\lambda_w = 0.01$ is equivalent to an exponential distribution with failure rate $\lambda = 0.01$.

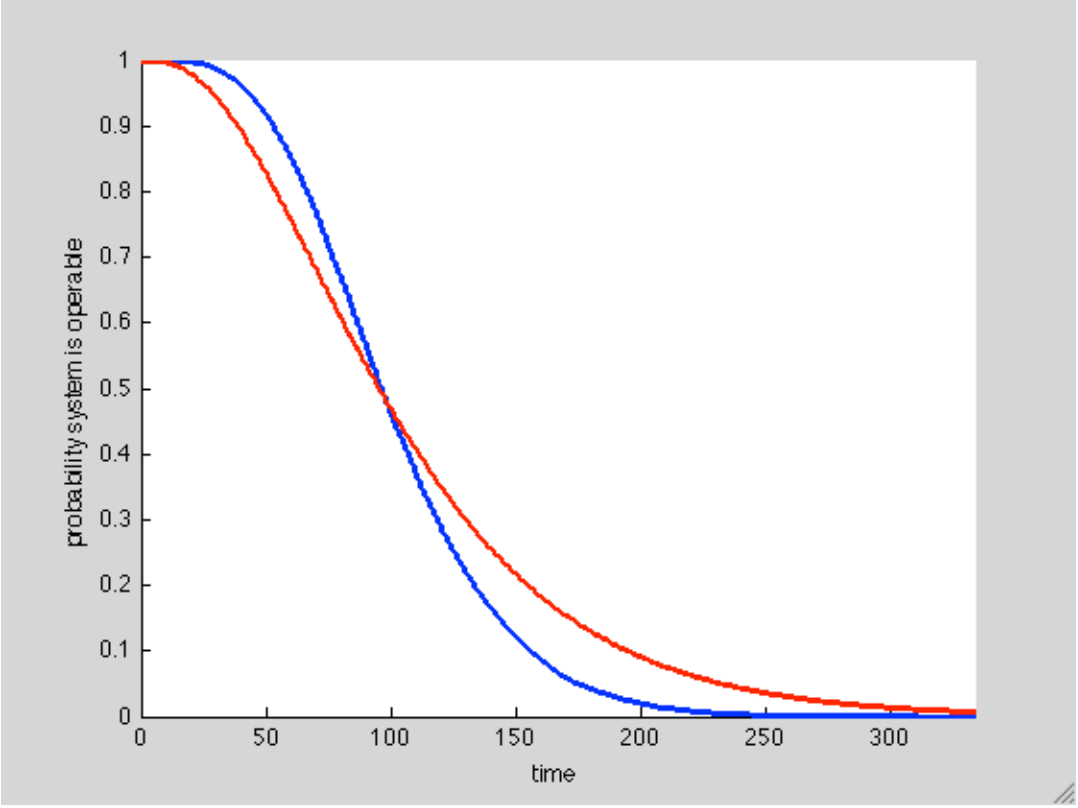


Figure 14 – Simulation of Weibull Failure Distribution in Blue, analytic result for corresponding Exponential Failure Distribution in Red.

Chapter 6: Auxiliary Feedwater System Example

6.1 Chapter Introduction

This Chapter will apply the modeling and simulation approach developed thus far to a more realistic problem involving an auxiliary feedwater system consisting of eleven components/subsystems as shown in Figure 15. This system was chosen for examination because it was extensively studied by Fleming, Mosleh, and others (Fleming, 1986, Mosleh et. al. 1989, Mosleh, 1991) at the time when current methods for conducting common cause failure analysis were developed. Analytic methods were used to develop a CCF model for this system, and therefore there is a body of existing work against which to compare the simulation method.

6.2 Description of the Auxiliary Feedwater System

The system is a back-up system intended to provide a continuous flow of cooling water to a downstream system if the normal cooling system fails. The system includes the following eleven components and subsystems:

- A storage tank and inlet valve feeding a common suction header (modeled as a single component)
- Three centrifugal pumps, two motor-driven, and one turbine-driven (modeled as six components including the pumps, motors, and turbine)
- Four motor-driven discharge valves, each feeding a downstream steam generator (modeled as four components). A schematic of the system is provided in Figure 15, and a reliability block diagram is provided in Figure 16. The system is

considered functioning if at least one pump is providing flow to at least two steam generators.

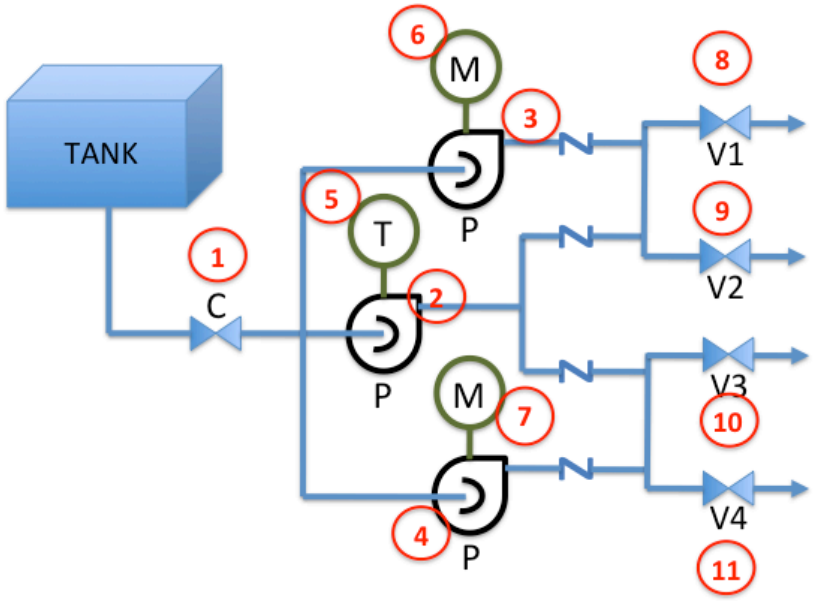


Figure 15 – Auxiliary Feedwater System studied by Fleming and Mosleh; numbers in circles correspond to component numbers in reliability block diagram

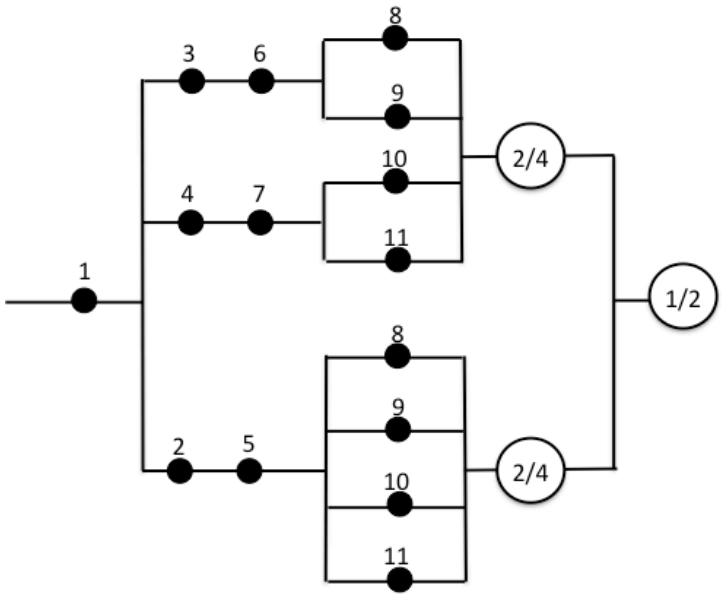


Figure 16 - Reliability block diagram for auxiliary feedwater system

This system can be represented by a System State Vector **x** with eleven elements as follows:

x_1 = 1 if the tank and inlet valve provide water to the pump inlet; 0 otherwise
x_2 = 1 if motor #1 is functioning; 0 otherwise
x_3 = 1 if motor #2 is functioning; 0 otherwise
x_4 = 1 if the steam turbine is functioning; 0 otherwise
x_5 = 1 if pump #1 is functioning; 0 otherwise
x_6 = 1 if pump #2 is functioning; 0 otherwise
x_7 = 1 if pump #3 is functioning; 0 otherwise
x_8 = 1 if motor operated valve #1 is functioning; 0 otherwise
x_9 = 1 if motor operated valve #2 is functioning; 0 otherwise
x_{10} = 1 if motor operated valve #3 is functioning; 0 otherwise
x_{11} = 1 if motor operated valve #4 is functioning; 0 otherwise

The structure function for the system is then:

$$\phi(\mathbf{x}) = 1 \text{ if } x_1*(x_2*x_5 + x_3*x_6 + x_4*x_7)*(x_8 + x_9 + x_{10} + x_{11} - 1) \geq 1;\ 0 \text{ otherwise}$$

6.3 Analytic Solution from the Literature:

Fleming (1986) provided the following analytic solution for the reliability of the auxiliary feedwater system:

$$
\begin{aligned}
R_{system} = 1 - {} & \\
& C \\
& + 4V_1^3 \\
& + (P_1 + T)(P_1 + M_1)(P_1 + M_1) \\
& + 4V_1 (P_1 + T)(P_1 + M_1) \qquad \Big\} \text{ Independent Terms} \\
& + V_4 + P_3 + 4V_3 \\
& + 12V_2V_1 + 15V_2^2 \\
& + 10P_1V_2(P_1 + T) + 10P_1V_2 M_1 \\
& + 10V_2P_2 + 10V_2M_1T \\
& + 3P_1P_2 + 3P_2^2 + 2P_2M_1 \qquad \Big\} \text{ Common Cause Terms} \\
& + 2P_2M_2 + 4P_2V_1 + P_2T \\
& + M_2P_1 + M_2T
\end{aligned}
$$

where the individual terms are the sum of a failure on demand probability and a probability of failure during run time as a function of t; for example $M_1 = \lambda_{1s} + \lambda_{1r}\, t$. The terms have the following definitions and values:

Term	Definition	Prob of failure on demand	Prob of failure during run time
C	Failure of tank or manual inlet valve	$\lambda_{SC} = 0$	λ_{RC} = 6.9E-8/hr.
M_1	Independent failure of one motor	λ_{SM1} = 1.49E-3	λ_{RM1} =1.61E-5/hr.
M_2	Common cause failure of both motors	λ_{SM2} = 2.98E-4	λ_{RM2} = 1.09E-6/hr.
P_1	Independent failure of one pump	λ_{SP1} = 8.68E-4	λ_{RP1} = 1.62E-5/hr.
P_2	Common cause failure of two pumps	λ_{SP2} = 2.17E-4	λ_{RP1} = 2.70E-7/hr.
P_3	Common cause failure of three pumps	λ_{SP3} = 5.21E-4	λ_{RP1} = 1.13E-6/hr.
T	Independent failure of turbine	λ_{ST} = 3.15E-2	λ_{RT} = 1.01E-3/hr.
V_1	Independent failure of one outlet valve	λ_{SV1} = 3.79E-3	$\lambda_{RV1} = 0$
V_2	Common cause failure of two outlet valves	λ_{SV2} = 4.49E-5	$\lambda_{RV2} = 0$
V_3	Common cause failure of three outlet valves	λ_{SV3} = 1.02E-5	$\lambda_{RV3} = 0$
V_4	Common cause failure of four outlet valves	λ_{SV4} = 3.80E-4	$\lambda_{RV4} = 0$

Table 8 - Parameter values for Auxiliary Feedwater System

6.4 Comparison of Analytic and Simulation Solutions:

Note that the analytic solution provided above is not exact - it makes use of the rare event approximation, which is reasonable in the context of the expected use of this model, but can introduce significant errors if used in a scenario when individual events are not rare. To estimate the error introduced by this approximation, a simulation was performed using the same general method as in the previous examples. Due to length, the MATLAB code is provided in the Appendix. For the auxiliary feedwater system, the practical concern is that the system will start on demand and continue to operate for a

period of up to a few days. In this case, the analytic solution is close to the simulation solution and is slightly conservative, as shown in Figure 17. However, if the analytic solution is extended over long periods of time, substantial error is introduced, because the events in question are no longer rare; over sufficient time, most of the components will fail. So for example, if this type of analytic approach were used to determine the expected lifetime of a system that is intended to run to failure, the answer would be wrong, and the simulation approach would be more likely to provide a credible answer, assuming the individual event rate parameters can be estimated with some degree of accuracy. A comparison of the analytic solution and the simulation solution extended over long system operating time is provided in Figure 18.

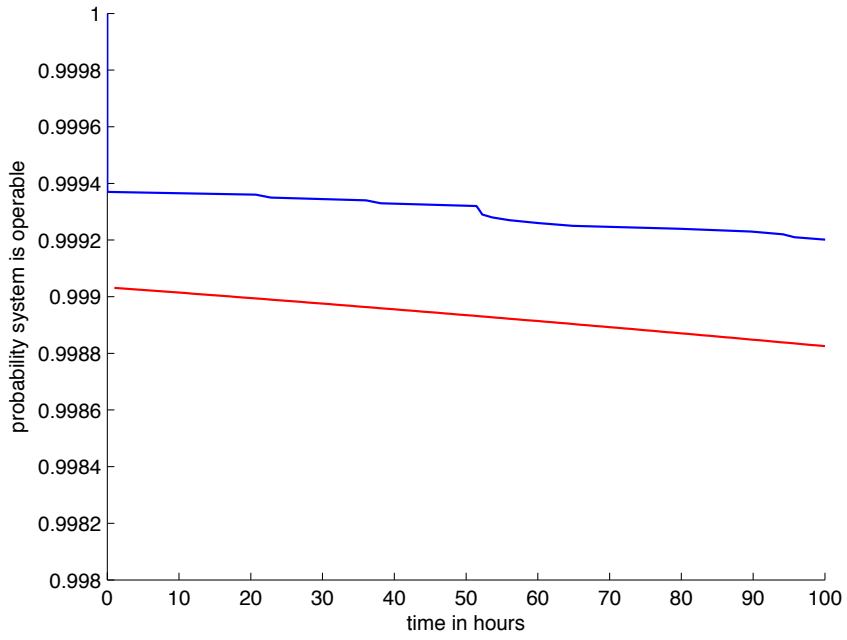


Figure 17 – Survival function for auxiliary feedwater system over the first 100 hours of system operation. Analytic result using the rare event approximation is shown in red, simulation result is shown in blue. 100,000 simulation runs. The rare event approximation is reasonable and slightly conservative.

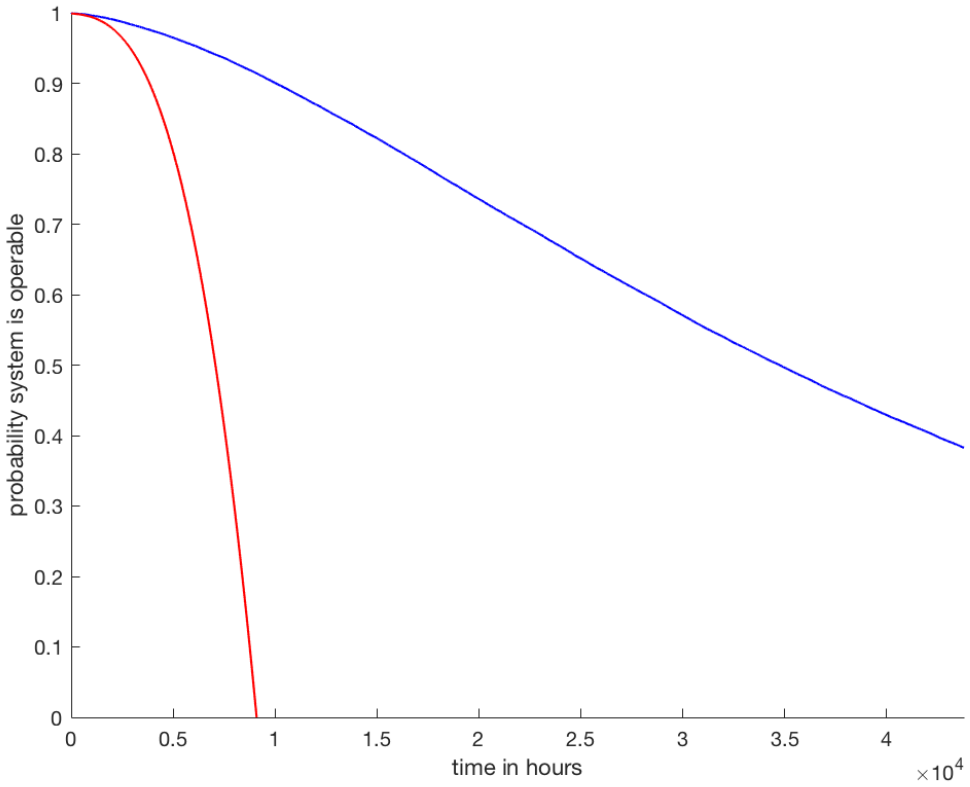


Figure 18 – Survival function for auxiliary feedwater system over five years of operation. The simulation is shown in blue. The analytic solution using the rare event approximation is shown in red, and is clearly wrong in this application since failure events are no longer rare over this extended time period. 100,000 simulation runs.

Chapter 7: Application to a New Class of Problems – Swarming Unmanned Vehicles

7.1 Chapter Introduction:

The advent of small and relatively inexpensive unmanned vehicles that are capable of hosting sophisticated yet low cost sensors has given rise to the concept of using large numbers of these vehicles operating cooperatively to quickly search large areas. These vehicles can be organized as heterogeneous teams to assist humans in performing complex search tasks more effectively and safely than using traditional methods. One example of research along these lines is the European Union's "ICARUS" project which is developing a family of unmanned vehicles with software and communications systems to support search and rescue operations (DeCubber, et. al., 2012). These operations are often conducted in hazardous environments where use of an unmanned system can avoid risk to humans.

The desire to operate these vehicles with a small number of remote human operators who may also be busy with other tasks, the desire to reduce decision and reaction time, and practical limitations of communications bandwidth, latency, and reliability, all lead to an imperative to make the individual vehicles as autonomous as possible, and to consider decentralized command and control approaches (Scheidt, 2015). Because these systems might behave in ways that are suggestive of groups of animals cooperating to achieve a task, there is a significant body of research on biologically-inspired approaches to autonomy and control (Steinberg, 2011). When these vehicles are self-organizing based on local autonomy and interactions with nearby neighbors, rather than centrally controlled, they are often referred to as swarms (Brambilla, et. al., 2013). Potential advantages of swarms include the ability to search a large area much more

quickly than with a single vehicle, and the potential for greater system reliability by avoiding single point failures, since the swarm is made up of many individuals, and since there is not a centralized hierarchical control system. If each vehicle is considered to be a redundant component in a highly parallel search system, and if vehicle failures are independent of each other, then the overall reliability of the system should be very high, with gradual and predictable degradation, giving confidence that the search can be completed within in a predictable amount of time. In the swarming literature, this attribute is sometimes referred to as "robustness" (Brambilla, et. al., 2013).

Although the potential advantages of unmanned vehicle swarms are intriguing, and although a substantial amount of academic research has been done to explore and develop a conceptual framework for swarming behavior, there are practical concerns that have slowed the emergence of this technology from the laboratory. Because swarms are not controlled by a rigid hierarchy, they can exhibit "emergent behaviors" which are group behaviors that arise from each vehicle optimizing its performance toward a goal, rather than from an explicit control algorithm. This is somewhat analogous to flocking behavior in birds. Modeling tools are being developed to understand and predict this behavior, for purposes of system design and evaluation (Lancaster and Gustafson, 2013). Additional tools that help designers understand the impact of component and system failures across multiple vehicles may accelerate the adoption of swarming approaches that would otherwise be inhibited by the lack of an extensive suite of systems engineering tools; unlike for the case of a single vehicle, the cost of "learning by doing" may be prohibitive for systems made up of many vehicles, even if the individual vehicles are affordable.

Tools for modeling independent and dependent failures of vehicles in a swarm can be used either as part of a system design process or as part of a search planning process. In design, issues of individual vehicle reliability can be addressed by means such as selection of reliable components and redundancy in critical subsystems. Resilience to common cause failures, on the other hand, might be best addressed through approaches such as introduction of deliberate heterogeneity in the design of the overall swarm, avoiding hardware and software "monocultures" which might lead to common cause failures. Design trades between system cost, complexity, maintainability, and overall robustness can be examined through the use of a simulation tool. In mission planning, a simulation tool can help answer questions such as the number of vehicles needed to reliably complete a mission in a specific amount of time, taking into account expected failure rates.

7.2 Dynamic Formation and Disbanding of Common Cause Groups

As noted in Chapter 1, the number of unique combinations of failures in a group of n identical components or vehicles is bounded by $\sum_{r=1}^{n} \binom{n}{r}$, where n is the number of components in a common cause group. For a swarming vehicle application, where n may be a very large number of vehicles, it isn't feasible to explicitly enumerate all of the different combinations of failures. Furthermore, if the common cause failure is due to environmental phenomena that are limited in geographic extent, or is due to proximity between vehicles, then it may be necessary to dynamically form common cause groups as individual vehicles move in and out of an area, or in and out of proximity to each other.

Figure 19 shows conceptually how a common cause group can be dynamically composed based on proximity to an initial failure, by generating additional common cause failures according to a function of distance from the locus of the failure event.

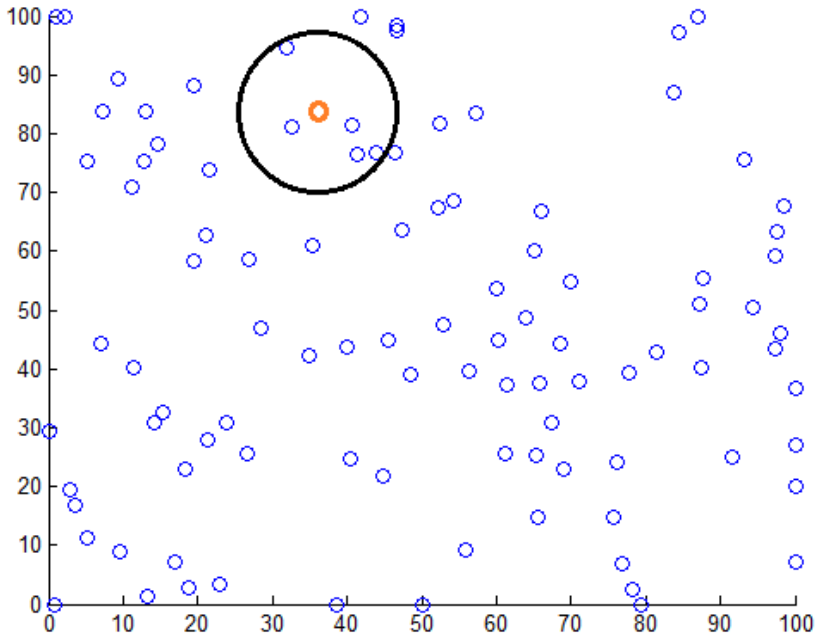


Figure 19 – Vehicle locations in the search box at the time of failure. A common cause group (circled) is dynamically formed around the vehicle that initiated the failure event, based on user-set parameters. Vehicles within a circle of specified radius fail with probability β, given failure of the initiating vehicle.

7.3 Development of a simulation model for the swarming vehicle problem

There are several potential challenges in applying discrete event simulation to modeling the reliability of a vehicle swarm: the potential dependency among failures must be taken into consideration in the model in a way that manages the explosion in combinations of failures as the value of *n* grows large; the modeling approach must be flexible to both temporal and spatial changes in event rates; and the definition of common

cause groups needs to be dynamic. In analyzing failures in an industrial plant, common cause groups can be statically defined based on common function, common design, or physical proximity. In a swarm, the functions of swarm members and their physical proximity can change over time.

The sample problem chosen to elucidate the simulation method is a swarm search problem, where a large number of vehicles are executing a random search pattern in a rectangular search area. Note that although a random search pattern was used here, a more deliberative search algorithm can be accommodated without changing the rest of the model; the model just needs to be able to calculate and report each vehicle's position at a specified time. Individual vehicle (independent) failures may be caused either by internal malfunctions of hardware or software, or by a command malfunction (for example, an external controller ordering a vehicle to fly into terrain), or by external environmental conditions such as weather. For modeling purposes, the single-vehicle failure rate can be calculated as a sum of independent failure rates attributed to each of these causes. A single-vehicle reliability model is a necessary building block for the overall swarm reliability model; fortunately there are well-developed tools for building reliability models for individual vehicles, and if the vehicles are relatively inexpensive it is possible to conduct enough real-world testing to validate these models and predict the single-vehicle failure rate with reasonable confidence.

The simplest model would assume that all types of single-vehicle failures are independent, and have a failure time adequately modeled by an exponential distribution with parameter λ. Therefore the single vehicle failure rate per each mode of failure is a constant value, λ and failures of the vehicle according to this mode can be described

using a homogeneous Poisson Process with rate λ. A more complicated model would allow for failure rates that can vary as a function of either time or geographic position or both. The time-variance in failure rate is intended to incorporate phenomena such as infant mortality and wear-out of vehicle components, while the spatial variation in failure rate is intended to incorporate environmental factors related to physical location that may contribute to failure. For example, in an unmanned ground vehicle scenario, a particular region might have more difficult terrain, causing a higher failure rate in that region. Time varying failure behavior will be modeled using a non-homogeneous Poisson process.

In the simulation developed herein, common cause failures across multiple vehicles are modeled as secondary "shock" events in which one or more additional vehicles fail simultaneously, in conjunction with the initial failure of a single vehicle. Potential root causes of these failures in a swarming vehicle scenario might include:

- Failure of command or navigation systems affecting multiple vehicles at once
- Software errors that are triggered nearly simultaneously on multiple vehicles due to some common triggering event
- Failures due to lack of availability of a communications path through a nearest neighbor
- Failures due to common environmental conditions in a region, such as wing icing.

In the sample problem, the susceptibility to common cause failure of each other vehicle in the swarm is determined by distance to the initial failure; in this way, the common cause groups are formed dynamically based on physical proximity, which is

physically intuitive for the common environmental and nearest neighbor types of common cause failures. For other types of failures, a common cause failure model based on "logical proximity" might be suggested.

The first step in the simulation-based analysis is to define the search problem in terms of numbers of vehicles *n*, size and shape of the search area, and some metrics for analysis, such as total area searched before system failure or time to system failure. In the context of a swarm, it is assumed that individual vehicles will fail frequently, so overall system failure occurs when the total number of vehicles in the swarm falls below some specified number, or when the total area searched prior to system failure or prior to a specified time is below some specified value. As noted above, one key element of a discrete event simulation is the tracking of system state as a function of time (Law and Kelton, 1991). During simulation, the failure criterion is formalized by writing a structure function $\phi(\mathbf{x})$ that determines whether the system is operable as a function of the states of the individual components. The state vector **x** is an array of length *n*, and represents the states of each of the vehicles in the swarm; the structure function $\phi(\mathbf{x})$ returns a value of one if the system is operable and zero if not operable. At the start of simulation, the state vector **x** is specified; in the example problem all of the vehicles are assumed operable at the start, so each element of **x** equals one, and the length of **x** is the initial number of vehicles, *n*. Alternatively, the simulation can be initiated with specified components failed, which is also a way to perform sensitivity analysis in a design process. A key task for the system designer is to develop clear definitions of system operability and failure which define the structure function; in other words, defining the possible combinations of operable vehicles that cause the overall system to be operable,

or conversely the possible sets of failed vehicles that causes the whole system to fail. In a system of identical vehicles without specialization of functions, this can be relatively simple: if the number of operable vehicles is above some minimum threshold, the whole system is considered operable. However, if the vehicles are specialized in design or function, this can be considerably more complex.

As in the previous examples, the simulation code uses random variates drawn from appropriate distributions to generate events at random times. These events can be either changes in velocity and direction of the vehicles as part of the random search pattern, single vehicle (independent) failures, common cause failures involving more than one vehicle, or the "null" event, meaning no significant event at that time step. The rationale for including the null event was explained above as part of the "thinning" process needed to accommodate temporal and spatial changes in event rates. As simulated events occur, the simulation tracks the positions of the vehicles within the search space and changes to the system state vector **x** caused by failure events. Changes are implemented by an element-wise multiplication of the previous system state vector **x** by a consequence vector **y** which contains zeros in the positions of vehicles which have failed due to the current random event, and ones elsewhere. This produces the new system state vector.

After each event, system functionality is evaluated using the structure function $\phi(\mathbf{x})$; if the system is still operable, time to subsequent events is generated randomly at a constant maximum event rate which is greater than the sum of all of the individual event rates evaluated across the time interval being considered in the simulation. If the evaluation of the structure function after an event indicates that the system is not

operable, the simulation stops and reports the time of failure and other relevant statistics such as total area searched or number of vehicle hours of search accomplished.

The MATLAB code for this simulation is provided in the Appendix. A large number of runs were conducted to exercise the model and develop a useful estimate of expected system performance under varying assumptions of the degree of vehicle dependence. Ordering all of the simulation runs by time to failure leads to a plot of the probability that the system is operable as a function of time. Additionally, the simulation tool can develop statistics on the total area searched prior to failure, which is a useful metric of the effectiveness of the swarm in performing a mission.

7.4 Simulation Parameters and Results:

The sample problem of a search using a large number of unmanned vehicles was exercised using the simulation technique given above. The simulation tool is flexible with regard to parameters such as search area, number of vehicles, and vehicle speed. However, to scope the problem and provide some reasonable numbers to use in the simulation, major search efforts from recent history were considered; for example, the search for Air France Flight 447, which crashed in the Atlantic in the early morning of 1 June 2009. There were multiple search phases conducted, starting with an air search in the immediate aftermath of the crash that covered a sea surface area of tens of thousands of square kilometers, and continuing with sea surface and undersea search efforts for almost two years, until the debris field was discovered and the black boxes were retrieved. The first acoustic search for "pinger" signals from the black boxes focused on a circular area 40 nautical miles in diameter, or approximately 4300 $km.^2$ Subsequent

searches informed by probabilistic analysis of the likely location of the debris field still had to cover areas of several thousand square kilometers of ocean bottom (Stone, et. al., 2014). Of course, every search problem is different, but this is indicative of the scale of problem being considered. If the search has an element of urgency, for example if it involves the potential of human survivors, an obvious strategy for accelerating the search is to increase the number of vehicles used, so that a given area can be searched more quickly.

For the example, simulation problem with results presented herein, the following specific simulation attributes were used:

(a) The search area was modeled as a rectangle with dimensions (Xmax and Ymax) = 100 x 100 km, for a total area of 10,000 km^2, consistent with a typical large area search problem like the Air France search.

(b) Number of vehicles (n) = 100. This is a user selected parameter, and the model has been exercised over a range of numbers of vehicles, primarily to show that it is scalable to large problems. Since small UAVs are relatively affordable compared to manned aircraft, use of 100 or more vehicles might be reasonable in the context of a major search effort.

(c) Vehicle speed is uniformly distributed from 50 to 100 km/hr, reflecting typical speeds of small unmanned air vehicles. Vehicles make course changes uniformly distributed in azimuth at random times according to a Poisson process, at a rate of 20 course changes per hour, and are constrained to stay inside the search box

(d) Individual vehicle failure rate (per hour) is expressed generally as:

$$\lambda = \alpha + \beta\,(t) + \gamma\,(x) + \delta\,(y)$$

where β, γ, and δ can represent any arbitrary function of time and location. For a geographically and time independent failure rate, the β, γ, and δ terms would reduce to zero and this would be a homogeneous Poisson process. In the particular example problem shown, the function $\lambda = a + bt^{1/3} + cx^{1/3} + dy^{1/3}$ was used, producing a failure rate that increases as a function of both time and distance from the origin. The values of *a* through *d* are user-selected parameters, and can be adjusted across a range of values to show the sensitivity of mission success to reliability. This function is clearly somewhat arbitrary, and is simply intended to illustrate that the modeling process can accommodate both spatially and temporally varying failure rates, using the "thinning" process described previously.

In analysis of a system of specific types of vehicles, the failure rate function should reflect physics-based or statistics-based insights on the actual failure rates as a function of time and location. For real systems, failure rate as a function of time typically decreases as vehicles with latent defects fail, while more reliable vehicles continue operating. However, failure rates often increase in later stages of life due to wear-out of components that are designed for finite lifetime. This combination of phenomena produces the so-called "bathtub curve" of failure rate as a function of time (Rausand and Hoyland, 2004). If the individual vehicle designs are sufficiently mature to allow collection of meaningful reliability data, then a failure rate function can be approximated from data using standard curve fitting methods. Focusing on the geographic variability in hazard rates, a more refined approach would be to develop a hazard "heat map" of the area in question based on some physical insight into the nature of the hazard. In this case, the γ(x) and δ(y) terms could be found from a look up table.

In the sample problem, common cause failures occur according to a shock process in which the failure of a single vehicle causes nearby vehicles to fail with probability β, where β is a function of distance to the initial failed vehicle. This is a method for dynamically creating common cause groups according to physical proximity. Conceptually, this dynamic grouping approach can also be extended to other measures of commonality which might change over time. The specific values of β were adjusted over a range to show the sensitivity of overall system performance to common cause failures.

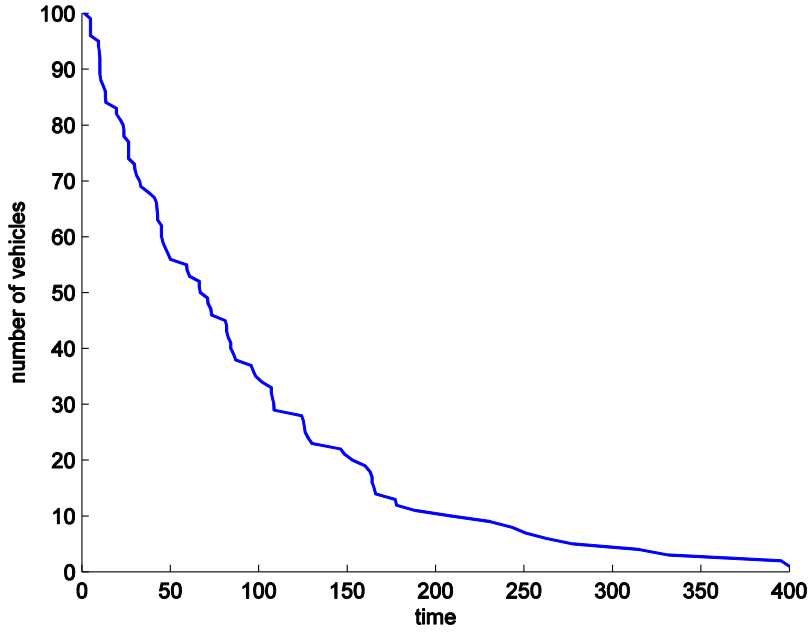


Figure 20 – Number of operable vehicles over time for a typical simulation run. Assuming a constant average search rate per vehicle, the searched area is proportional to the area under the curve.

Figure 20 shows a plot of number of operable vehicles over time for a typical simulation run. For each vehicle, the search rate in km^2 per hour is approximately the average speed multiplied by sensor swath width, where swath width is related to the maximum range at which the sensor can detect an object of interest with some specified probability of

detection. Search area per vehicle is search rate multiplied by time. Since the x axis is time and the y axis is number of operable aircraft, the area under the curve is approximately proportional to the total area searched, noting that in a random search pattern some areas will be searched more than once. Therefore, area under the curve is a reasonable metric for overall mission performance. It is also evident that a single simulation run has quite a bit of "noise" due to random variability. Therefore multiple simulation runs should be accomplished to develop metrics that are statistically bounded within an acceptable error range. Figure 21 shows the performance of the swarm using the same failure rate parameters averaged over 50 runs. To examine the potential impact of common cause failures, the parameter β was adjusted in increments from zero to 0.6. Figure 22 shows the impact of common cause failures on vehicle population and overall search area as a function of β.

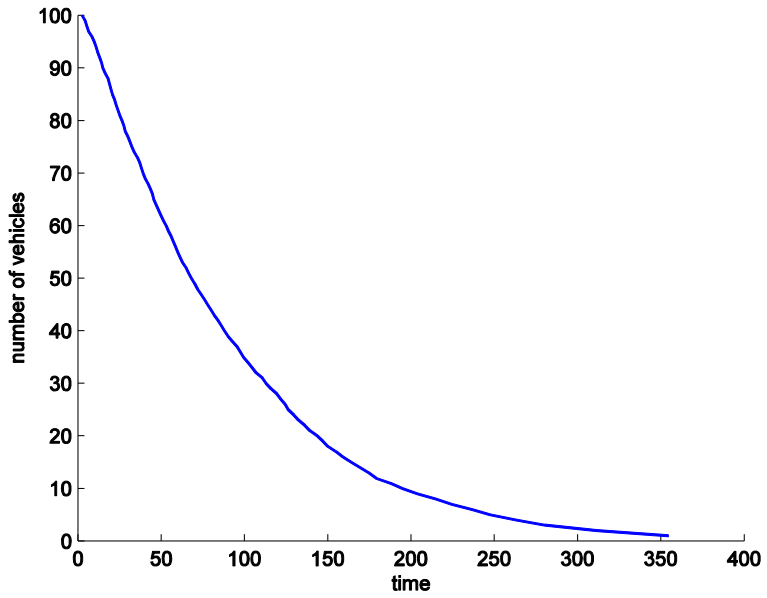


Figure 21 – Number of operable vehicles over time averaged over 50 simulation runs. Searched area is approximately proportional to the area under the curve.

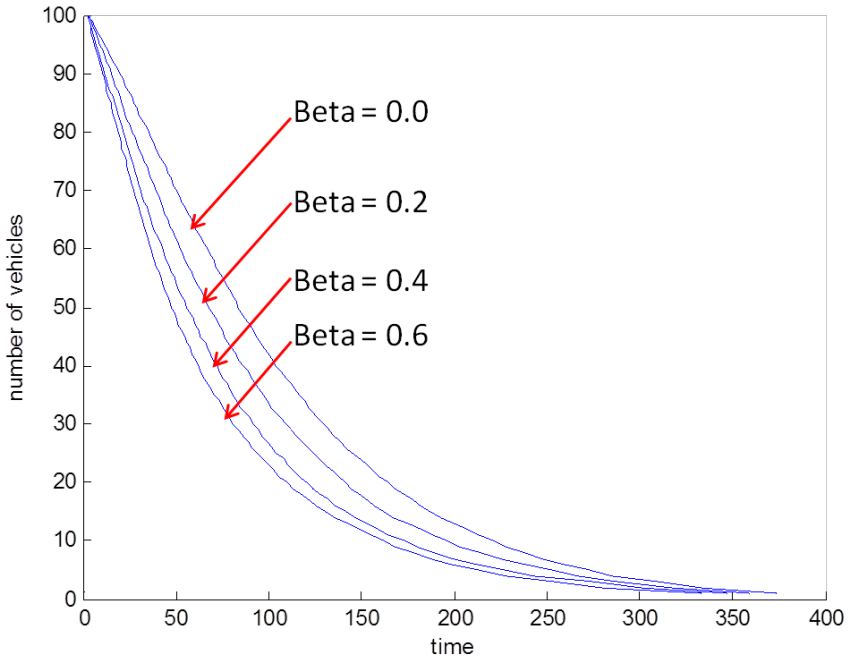


Figure 22 – Effect of common cause failures; curve gets closer to the origin as the common cause failure parameter Beta is increased. Total searched area is approximately proportional to the area under the curve.

Clearly, higher values of β, indicating more common cause failures, lead to reduced overall system performance. This raises the question of what value to select for β? Hockstad and Rausand (2008) describe several procedures for estimating β that have been developed for nuclear power plant and offshore oil applications. These procedures are typically based on expert judgments of the degree to which a design incorporates features that make it more or less susceptible to common cause failures. For example, Humphrey's method establishes eight sub-factors that are scored to assess susceptibility to common cause failures, and then entered in a formula to provide an estimate of β ranging from 0.01 to 0.3. Examples of the factors are the degree of design similarity, degree of physical similarity, amount and rigor of testing, and so on. Many of the design principles intended to reduce β, such as introducing design heterogeneity between

redundant components and eliminating paths for operator error to simultaneously impact multiple components, would also seem to be applicable to the design of collaborative teams of unmanned vehicles, but it is unclear if the numerical values developed using these scoring methods are valid, since they were developed in very different application domains. The development of a comprehensive approach to estimating and predicting the impact of common cause failures in the nuclear power industry required a major effort over about two decades; a comparable effort may be needed to provide similarly useful tools for evaluating the impact of common cause failures on large swarms of autonomous vehicles.

More generally, how would the reliability model developed in this chapter be used in a design or mission planning process? In design, reliability can be considered both an input and an output: starting with a design reference mission such as searching a given area in a specified amount of time, the model can be exercised to determine the number of required vehicles to accomplish the mission under different assumptions about vehicle reliability, forming a technical basis for a minimum reliability specification. Later in the design process, the reliability model can be used to predict overall system performance based on reliability testing at the component or single vehicle level. In a mission planning process, the simulation might help planners decide how many vehicles to use for a given search based on size of area to be searched and desired time to complete the search.

7.5 Chapter Summary

This Chapter developed a flexible simulation approach that can be used for modeling the performance of an unmanned vehicle swarm under different assumptions about vehicle reliability, considering both independent and common cause failures. Model validation ultimately requires experimentation and data collection with an actual swarm; however, this is not typically possible in a design process when the vehicles are still in early prototype stage, have not been produced in quantities necessary for robust swarm experiments, and aren't necessarily representative of production vehicles. However, the model can be very useful in translating an overall system performance goal into individual vehicle reliability requirements, and in doing trade studies on alternative technical approaches. For example, in choosing between smaller numbers of vehicles with highly reliable components and more numerous, less expensive, but less reliable vehicles, the model can provide useful insights to the system engineer.

Because discrete event simulation techniques rely on random variates drawn from appropriate distributions, they are not "exact" answers; however, exact analytic solutions are not practical for the class of problems at hand, and with a sufficient number of simulation runs discrete event simulation can bound the "exact" solution within acceptable error bounds. This leads to questions of run time and practicality. For the sample problem provided in this paper, run times are on the order of a few seconds per simulation run on an ordinary personal computer, so producing large numbers of runs is not impractical.

This particular simulation focused on a homogeneous fleet of expendable vehicles. Future work in this area might extend the model to systems where the vehicles

are heterogeneous either in form or in function, where the vehicles are repairable, or where additional vehicles are injected into the search at appropriate times. Validation of the model through mixed hardware/software simulations using a combination of real and virtual vehicles would also provide insight into the utility of the tool for actual design and search planning problems.

Chapter 8 – Summary, Conclusions, and Future Research

8.1 Chapter Introduction

The prior chapters described a process for using discrete event simulation to solve complex reliability problems in the presence of common cause failures. It was shown that this method overcomes some of the simplifying assumptions associated with earlier analytic approaches, and is flexible to a wide variety of problems. This chapter addresses some potential concerns and limitations associated with the simulation approach and identifies some potential future directions for research.

8.2. Usefulness of the Approach: Considerations of Scalability and Run Time

In assessing the utility of a simulation-based approach, a significant concern is whether the approach will scale gracefully to more complex problems. To explore this question, data was collected for a few sample problems on simulation run time as a function of number of runs, number of random events generated, and number of components in the system. The simulations were run on two different computers: an inexpensive Dell laptop purchased in 2010 running Windows 7, and an iMac purchased in 2009 running OS X. Run times are approximately the same on both machines. Results for the iMac are shown in Figure 23. As can be seen, the run times for all three examples are relatively short, typically several seconds. For example, 5,000 runs of the auxiliary feedwater system simulation with tmax = 10,000 hours produced approximately 232,000 random events, and ran in slightly under 8 seconds. For any particular problem, the run time is approximately proportional to the number of random events generated, which

increases with both the number of simulation runs and the length of the time period evaluated, from t=0 to t = tmax. The eleven component auxiliary feedwater system simulation has a steeper slope in terms of run time per number of random events, which seems reasonable since the complexity of the calculation performed at each random event increases with the number of components. However, the increase from three to five to eleven components does not cause a dramatic slowdown in run time, indicating that scaling to larger and more complex problems should not lead to unacceptably long run times.

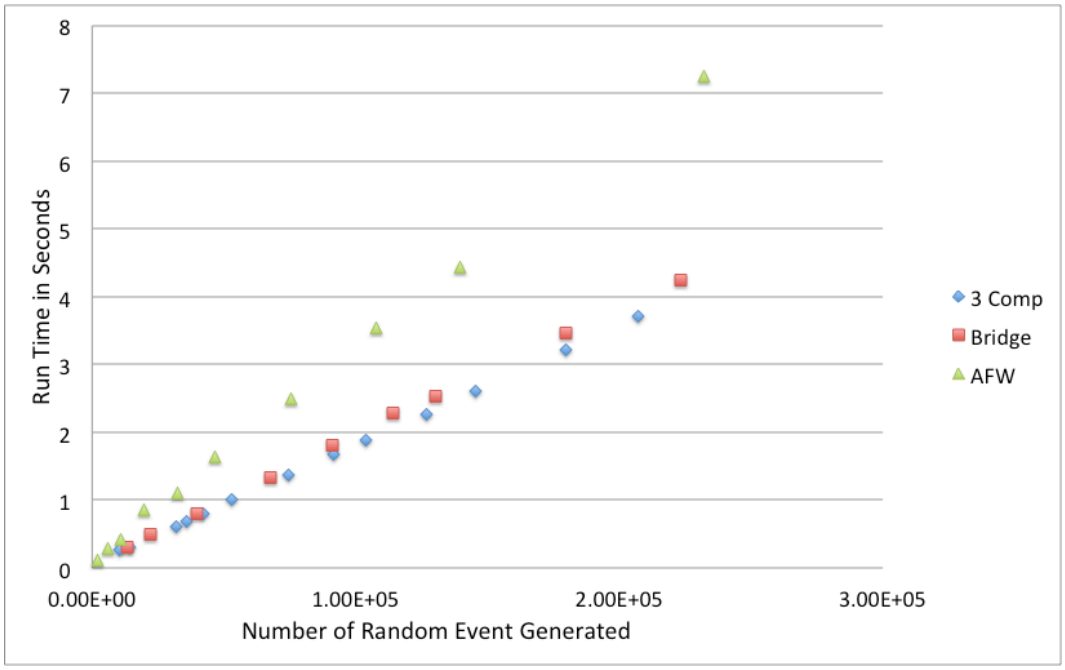


Figure 23 – Simulation run time as a function of number of random events generated

An important question in any simulation is "how many runs are enough?" A simple and intuitive approach is to start with a small number of runs to ensure the simulation is running correctly, and then increase the number of runs until the solution appears to converge. For the examples given, a good "eyeball" convergence typically

occurs somewhere between 1,000 and 10,000 runs. A more rigorous answer follows. Assume that the exact survivor function for the system is θ(t), and that the simulation produces an unbiased estimate of the survivor function, X(t). The assumption of an unbiased estimate is equivalent to stating that the simulation should converge toward the exact solution for the survivor function, given a sufficient number of simulation runs (Devore, 1991; Roberts, 2010):

$$\theta(t) \cong X(t) = \frac{1}{n}\sum_{i=1}^{n} X_i\,(t)$$

Note that in any single run, for a specified value of t, $X_i(t)$ equals one if t is less than the survival time generated by the simulation during that run, and zero if t is greater than the simulated survival time. So for example, if in 10,000 simulation runs there are 8,240 runs where the simulated survival time is greater than 100 hours, then the estimated survival function evaluated at t=100 is X(t) = 82.4%. The sample variance S^2 and approximate mean squared error (MSE) are defined as (Devore, 1991; Roberts, 2010) :

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - X)^2$$

$$MSE \cong S^2/n$$

For reasonably large values of *n*, the central limit theorem can be applied:

$$P\{|X - \theta| < cS/\sqrt{n}\} \cong 2\phi(c) - 1$$

where ϕ(c) is the standard Gaussian cumulative distribution function. For a value of c=1.96, corresponding to a 95% confidence interval, upper and lower bounds on the exact survival function θ(t) can be expressed as:

$$P\left\{X - 1.96\frac{S}{\sqrt{n}} < \theta < X + 1.96\frac{S}{\sqrt{n}}\right\} \cong 0.95$$

Evaluation of the upper and lower confidence bounds in MATLAB is fairly straightforward. Figure 24 shows implementation of this approach for example problem of the "Bridge" system described earlier. It can be seen that the upper and lower confidence limits become closer to the sample mean as the number of runs *n* is increased; a suitably high value of *n* can therefore be selected to provide the desired level of confidence in the simulation answer.

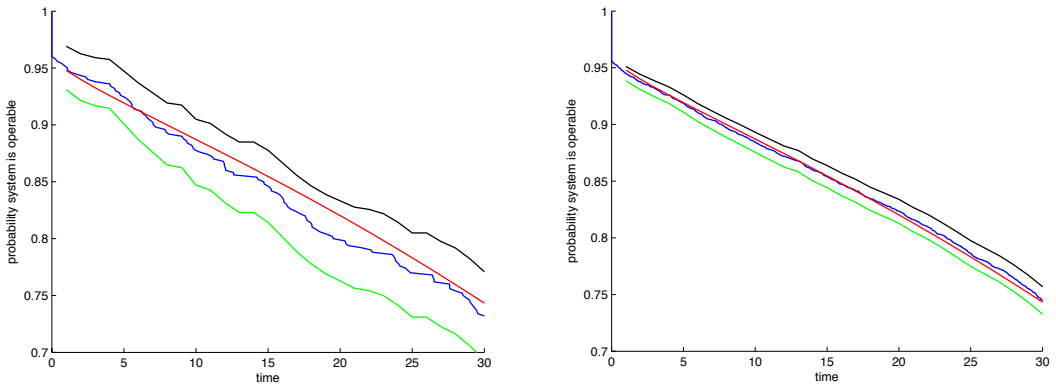


Figure 24 – Simulation of "Bridge" system; Analytic solution is shown in red, with simulation result in blue. 95% confidence limits are shown in black and green. Simulation on the left was done with 500 runs. Simulation on the right was done with 5,000 runs.

8.3 Summary and Conclusions

The goal of this study was to develop extensions to the basic parameter common cause failure model that would allow incorporation of non-constant failure rates, and to demonstrate a practical simulation-based approach for determining the survival function of relatively complex systems where implementation of an exact analytic solution is difficult or impractical. Since the simulation approach does not require the rare event

approximation, it converges to the exact solution for the survivor function within small error bounds, given a sufficient number of simulation runs. The simulation uses the structure function directly to evaluate system operability, making it relatively straightforward to implement from a reliability block diagram; development of a fault tree is not explicitly required, although a fault tree may be helpful in developing a complete list of possible events, which is required for developing the event matrix **E**. The simulation method requires individual component failure rates and common cause event rates, which can be estimated using the same parameter estimation techniques that would be used in a fault tree analysis.

It was shown that for systems that are moderately complex, such as the auxiliary feedwater system, the simulation-based approach can avoid some problems that arise with traditional fault-tree based methods:

(a) The simulation method avoids the need to conduct an in-depth analysis of all possible cut sets including the many additional cut sets introduced by potential common cause failures; this reduces the potential for either overlooking or double counting some of the terms in the solution.

(b) The rare-event approximation is not required, avoiding a source of error that can lead to significantly wrong answers in cases where the events are not truly rare. This makes the simulation method attractive, particularly for analyses that involve non-repairable systems that are intended to run to failure.

(c) The simulation approach can deal with time-varying event rates, dynamic formation of common cause groups, and is scalable to highly parallel systems with large numbers of common cause groups.

Some typical concerns about simulation-based approaches and rebuttals to those concerns are:

(a) Simulation results are not exact. This is true, but results that are reliably within required statistical error bounds can be achieved by increasing the number of runs.

(b) Simulation results are not transparent; simulation is a "black box." This can be true, particularly when proprietary software is used and the internal workings of the simulation software can't readily be inspected. This is one of the reasons for using a general-purpose mathematical analysis tool such as MATLAB, since the code is readily inspectable and can be reproduced or altered to suit the needs of the analyst.

(c) Simulation requires specialized personnel with unique expertise and access to unique software. Hopefully, the example problems shown in this paper illustrate that simulation-based approaches are not overly complex, and should be accessible to a broad spectrum of engineers and analysts.

(d) Simulation-based methods are time consuming and resource-intensive. Development of the general simulation approach was somewhat time-consuming; however, extending the method to new problems such as the auxiliary feedwater system is relatively straightforward. Most of the effort involved is work that would also be required for an analytic solution, such as developing a complete list of all the possible failure events and their consequences, and developing a structure function that relates component reliability to system reliability. The

actual coding for a new problem of similar complexity takes several hours, and once developed, the model runs quickly on an ordinary personal computer.

In conclusion, the discrete event simulation method developed herein is straightforward to implement for real systems, and may be a useful alternative approach to modeling common cause failures in a number of cases where an analytic solution is too complex to implement or would require simplifying assumptions that are potentially incorrect. Once developed, a simulation model of the type described can be used either as a design tool or as an analysis tool. For example, suppose that a non-repairable system is being designed and has a threshold requirement for 90% probability of survival to particular operating time. Given a proposed system configuration, a reliability block diagram can be developed and used to derive the system structure function. The relationships between component failure rates, common cause event rates, and overall system survival can then be modeled. If the components being used are based on existing designs with known reliability metrics, the simulation can be used to compare different configurations and levels of redundancy to determine if the system reliability requirement can be met. Alternatively, the model can be used to develop requirements for component reliability to drive component-level design and testing.

Potential areas for future research to extend this work include application to practical problems and validation against real world data, as well as further studies of run time, convergence, and scalability to larger problems. This method may also be extensible to simulation of ageing phenomena where components can accumulate damage via shock events prior to failure. This is a topic of current interest (Anastasiadis, et. al., 2010), and is likely to require a simulation-based solution.

References

Anastasiadis S., Arnold R, Chukova S, and Hayakawa Y. 2010. Failure model for multicomponent systems with damage accumulation, in 'Advanced Reliability Modeling IV: Beyond the Traditional Reliability and Maintainability Approaches', McGraw-Hill, Taipei, pp. 9–16.

Apostolakis G, Moieni P., 1987, The Foundations of Models of Dependence in Probabilistic Safety Assessment. Reliability Engineering; 18: 177-95. doi: 10.1016/0143-8174(87)90097-7

Apostolakis, George E. "How useful is quantitative risk assessment?." *Risk analysis* 24.3 (2004): 515-520.

Atwood CL., 1987 The Binomial Failure Rate Common Cause Model. Technometrics; 28:139-48. doi: 10.1080/00401706.1986.10488115

Beal, Alasdair N. "A history of the safety factors." *Structural Engineer* 89.20 (2011): 20-26.

Brambilla, M., Ferrante, E., Birattari, M., & Dorigo, M., 2013, Swarm robotics: a review from the swarm engineering perspective. Swarm Intelligence; 7(1), 1-41. doi: 10.1007/s11721-012-0075-2

Carlisle, Rodney P. "Probabilistic risk assessment in nuclear reactors: engineering success, public relations failure." *Technology and culture* (1997): 920-941.

Collet J., 1996, Some Remarks on Rare-Event Approximation. IEEE Transactions on Reliability; 45: 106-8. doi: 10.1109/24.488924

David, Florence Nightingale. "Games, gods and gambling: The origins and history of probability and statistical ideas from the earliest times to the Newtonian era." (1962).

De Cubber, G., Doroftei, D., Baudoin, Y., Serrano, D., Chintamani, K., Sabino, R., & Ourevitch, S., 2012, ICARUS: Providing Unmanned Search and Rescue Tools. In 6th IARP Workshop on Risky Interventions and Environmental Surveillance (RISE), Warsaw, Poland.

Devore JL. Probability and Statistics 3rd edition. Pacific Grove, CA: Brooks/Cole Publishing Company; 1991.

Epstein S, Rauzy A, 2005, Can we trust PRA? Reliability Engineering and System Safety; 88: 195-205. doi:10.1016/j.ress.2004.07.013

Ericson, Clifton A., and Clifton Ll. "Fault tree analysis." *System Safety Conference, Orlando, Florida*. 1999.

Fleming KN. A reliability model for common mode failures in redundant safety systems. ReportGA-A13284, General Atomic Company, San Diego, CA, 1975.

Fleming KN, Mosleh A, Deremer RK., 1986, A systematic procedure for the incorporation of common cause events into risk and reliability models. Nuclear Engineering and Design; 93:245–79. doi: 10.1016/0029-5493(86)90223-2

Freudenburg, William R. "Perceived risk, real risk: Social science and the art of probabilistic risk assessment." *Science(Washington)* 242.4875 (1988): 44-49.

Gaertner, John, Ken Canavan, and Doug True. "Safety and operational benefits of risk-informed initiatives." *An EPRI White Paper, Electric Power Research Institute.* February 2008

Hockstad P, Rausand M., 2008, Common Cause Failure Modeling: Status and Trends. In: Handbook of Performability Engineering, Misra KB (ed). Springer: London, 2008; 621-40. doi: 10.1007/978-1-84800-131-2_39

Hoepfer VM, Saleh JH, Marais KB. On the value of redundancy subject to common-cause failures: Toward the resolution of an on-going debate. Reliability Engineering and System Safety 2009; 94:1904–16.

Keller W, Modarres M, 2005, A historical overview of probabilistic risk assessment development and its use in the nuclear power industry: a tribute to the late Professor Norman Carl Rasmussen, Reliability Engineering and System Safety; 89: 271-85. doi: 10.1016/j.ress.2004.08.022

Kolmogorov, Andreĭ Nikolaevich. "Foundations of the Theory of Probability." (1950).

Kumamoto H, Henley E. Probabilistic Risk Assessment for Engineers and Scientists, 2nd ed. New York: IEEE Press; 1996.

Lancaster, JP, Gustafson, DA, 2013, Predicting the behavior of robotic swarms in search and tag tasks. Procedia Computer Science, 20, 77-82. doi:10.1016/j.procs.2013.09.242

Law AM, Kelton DW. Simulation Modeling and Analysis, 2nd edition. New York, NY: McGraw Hill; 1991.

Leemis LM, Park SK. Discrete Event Simulation: A First Course. Upper Saddle River, NJ: Pearson Education; 2005.

Leemis LM. Variate Generation in Reliability. In: Faulin J, Juan AA, Martorell S, Ramirez-Marquez J, eds. Simulation Methods for Reliability and Availability of Complex Systems. London: Springer, 2010, p. 85-103.

Lewis PAW, Shedler GS, 1979, Simulation of nonhomogeneous Poisson processes by thinning. Naval Logistics Quarterly; 26: 403-13. doi: 10.1002/nav.3800260304

Li, Huiling. *Estimation of the Parameters of Marshall-Olkin Type Distribution Under Various Life-testing Scenarios*. Diss. George Washington University, 2005.

Littlefield S, Mazzuchi T, & Sarkani S, 2012, Predicting reliability in design of complex systems with common-cause failures and time-varying failure rates. In Systems Conference (SysCon), 2012 IEEE International. doi: 10.1109/SysCon.2012.6189455

Marshall AW, Olkin I, 1967, A multivariate exponential distribution. Journal of the American Statistical Association; 62:30–44. doi: 10.1080/01621459.1967.10482885

Mathworks Product Documentation, accessed 9 April 2016: http://www.mathworks.com/help/matlab/math/random-numbers-in-matlab.html

Matsumoto, Makoto, and Takuji Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator." *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998): 3-30.

Metropolis, Nicholas, and Stanislaw Ulam. "The monte carlo method." *Journal of the American statistical association* 44.247 (1949): 335-341.

Metropolis, Nicholas. "The beginning of the Monte Carlo method." *Los Alamos Science* 15.584 (1987): 125-130.

Mosleh, A., et al. *Procedures for treating common cause failures in safety and reliability studies: analytical background and techniques*. No. NUREG/CR-4780-Vol. 2; EPRI-NP-5613-Vol. 2; PLG-0547-Vol. 2. Nuclear Regulatory Commission, Washington, DC (USA). Div. of Systems Research; Electric Power Research Inst., Palo Alto, CA (USA); Pickard, Lowe and Garrick, Inc., Newport Beach, CA (USA), 1989.

Mosleh A. Common Cause Failures: An Analysis Methodology and Examples. Reliability Engineering and System Safety 1991; 34: 249-92.

Nance, Richard E. *A history of discrete event simulation programming languages*. ACM, 1996.

Rasmussen, Norman C. *Reactor Safety Study: An Assessment of Accident Risks in US Commercial Nuclear Power Plants: Accident Definition and Use of Event Trees*. US Atomic Energy Commission, 1974.

Rausand M, Hoyland A, 2004, System Reliability (2nd edn). John Wiley and Sons, Inc.: Hoboken NJ.

Rauzy A, 2008, Binary Decision Diagrams for Reliability Studies. In: Handbook of Performability Engineering, Misra KB (ed). Springer: London; 381-96. doi: 10.1007/978-1-84800-131-2_25

Roberts WJ. Discrete Event Simulation. Unpublished class notes, 2010.

Ross SM, 1993, Introduction to Probability Models (5th edn). Academic Press: San Diego, CA.

Sagan SD. The problem of redundancy problem: why more nuclear security forces produce less nuclear security. Risk Analysis 2004; 24(4): 935–46.

Scheidt, DH, 2015, Command and Control of Autonomous Unmanned Vehicles. In Handbook of Unmanned Aerial Vehicles (pp. 1273-1298). Springer Netherlands.

Smith, Anthony M., and Ian A. Watson. "Common cause failures—a dilemma in perspective." *Reliability Engineering* 1.2 (1980): 127-142.

Stamatelatos M, Homayoon D. Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners, 2nd edition. Washington, DC: National Aeronautics and Space Administration, NASA/SP2011-3421; December 2011.

Steinberg, M., 2011, Biologically-inspired approaches for self-organization, adaptation, and collaboration of heterogeneous autonomous systems. In SPIE Defense, Security, and Sensing (pp. 80620H-80620H). International Society for Optics and Photonics.

Stone, LD, Keller, CM, Kratzke, TM, & Strumpfer, JP, Search for the Underwater WQreckage of Air France Flight AF 447, 2014, Statistical Science, 29, no. 1, 69--80. doi:10.1214/13-STS420.

Styan, George PH. "Hadamard products and multivariate statistical analysis." *Linear algebra and its applications* 6 (1973): 217-240.

U.S. Government Accountability Office. Nuclear Regulatory Commission: Natural Hazard Assessments Could Be More Risk-Informed. GAO-12-465: Published: Apr 26, 2012. Publicly Released: May 29, 2012.

U.S. Nuclear Regulatory Commission. Probabilistic Risk Assessment. Fact Sheet; October 2007.

U.S Nuclear Regulatory Commission. An Approach for Determining the Technical Adequacy of Probabilistic Risk Assessment Results for Risk-Informed Activities, Regulatory Guide 1.200; March 2009.

Vesely WE, 1977, Estimating common cause failure probabilities in reliability and risk analyses: Marshall-Olkin specializations. Nuclear Systems Reliability Engineering and Risk Assessment. Society for Industrial and Applied Mathematics: Philadelphia.

Appendix – MATLAB Code

SECTION A1 – AUXILIARY FEEDWATER SYSTEM SIMULATION

```
% SIMULATION OF 11 COMPONENT AUX FEEDWATER SYSTEM

clear all % clears MATLAB workspace
clc % clears command window
format compact % fits more text in the command window
tic % starts timer for CPU run time

% INPUT MODEL PARAMETERS, TIME PERIOD, AND NUMBER OF SIMULATION RUNS

InitialX = ones (1,11); % All components initially undamaged; however,
can
                        % adjust values to allow pre-existing damage

tmax = 43800; % Input maximum time of interest

n=100000; % Number of runs

% Input Basic Parameters

Lambda_S_Tank = 0; % Prob of damage Tank or inlet valve piping at start
Lambda_S_PumpSingle = 8.68E-4; % Prob of independent pump damage at
start
Lambda_S_PumpDouble = 2.17E-4; % Prob of 2 pump common cause damage at
start
Lambda_S_PumpTriple = 5.21E-4; % Prob of 3 pump common cuase damage at
start
Lambda_S_Turbine = 3.15E-2; % Prob of turbine damage at start
Lambda_S_MotorSingle = 1.49E-3; % Prob of independent motor damage at
start
Lambda_S_MotorDouble = 2.98E-4; % Prob of 2 motor common cause damage
at start
Lambda_S_ValveSingle = 3.79E-3; % Prob of independent valve damage at
start
Lambda_S_ValveDouble = 4.49E-5; % Prob of 2 valve CC damage at start
Lambda_S_ValveTriple = 1.02E-5; % Prob of 3 valve CC damage at start
Lambda_S_ValveQuad = 3.80E-4; % Probe of 4 valve CC damage at start

LambdaR_Tank = 6.90E-8; % Tank or valve damage rate during run - scale
factor
AlphaR_Tank = 1; %  Tank damage rate during run - shape factor

LambdaR_PumpSingle = 1.62E-5; % Pump damage rate during run - scale
factor
AlphaR_PumpSingle = 1; %  Pump damage rate during run - shape factor

LambdaR_PumpDouble = 2.70E-7; % 2 Pump CC damage rate during run -
scale factor
AlphaR_PumpDouble = 1; %  2 Pump CC damage rate during run - shape
factor
```

```
LambdaR_PumpTriple = 1.13E-6; % 3 Pump CC damage rate during run -
scale factor
AlphaR_PumpTriple = 1; %  3 Pump CC damage rate during run - shape
factor

LambdaR_Turbine = 1.01E-3; % Turbine damage rate during run - scale
factor
AlphaR_Turbine = 1; % Turbine damage rate during run - shape factor

LambdaR_MotorSingle = 1.61E-5; % Motor damage rate during run - scale
factor
AlphaR_MotorSingle = 1; %  Motor damage rate during run - shape factor

LambdaR_MotorDouble = 1.09E-6; % 2 Motor CC damage rate during run -
scale factor
AlphaR_MotorDouble = 1; %  2 Motor CC damage rate during run - shape
factor

LambdaR_ValveSingle = 0; % Valve damage rate during run - scale factor
AlphaR_ValveSingle = 1; %  Valve damage rate during run - shape factor

LambdaR_ValveDouble = 0; % 2 Valve CC damage rate during run - scale
factor
AlphaR_ValveDouble = 1; %  2 Valve CC damage rate during run - shape
factor

LambdaR_ValveTriple = 0; % 3 Valve CC damage rate during run - scale
factor
AlphaR_ValveTriple = 1; %  3 Valve CC damage rate during run - shape
factor

LambdaR_ValveQuad = 0; % 4 Valve CC damage rate during run - scale
factor
AlphaR_ValveQuad = 1; %  4 Valve CC damage rate during run - shape
factor



% Choose between random damage and fatal damage during start and run
% Set the following equal to one for random damage, otherwise damage
will
% be fatal

Rand_Tankstart = 0;
Rand_Pumpstart = 0;
Rand_Pumpstart2 = 0;
Rand_Pumpstart3 = 0;
Rand_Turbinestart = 0;
Rand_Motorstart = 0;
Rand_Motorstart2 = 0;
Rand_Valvestart = 0;
Rand_Valvestart2 = 0;
Rand_Valvestart3 = 0;
Rand_Valvestart4 = 0;

Rand_Tankrun = 0;
```

```
Rand_Pumprun = 0;
Rand_Pumprun2 = 0;
Rand_Pumprun3 = 0;
Rand_Turbinerun = 0;
Rand_Motorrun = 0;
Rand_Motorrun2 = 0;
Rand_Valverun = 0;
Rand_Valverun2 = 0;
Rand_Valverun3 = 0;
Rand_Valverun4 = 0;

epsilon = 0.5; % damage threshold; component values in X below
               % threshold indicate component failure


timevector = zeros (1,n); % A vector which collects the failure
                          % time in each run

% SET INITIAL E, DTV, P, MER:

E = ones (28,11);
% E is the event matrix.  Each row is a possible damage event

DTV = ones (1,11) *epsilon; %damage threshold vector

P = zeros(1,27);% Probability vector for failure on demand

P(1) = Lambda_S_Tank; % Tank damage
P(2) = Lambda_S_PumpSingle; % Indep damage to pump #1
P(3) = Lambda_S_PumpSingle; % Indep  damage to pump #2
P(4) = Lambda_S_PumpSingle; % Indep damage to pump #3
P(5) = Lambda_S_Turbine; % Indep damage to steam turbine
P(6) = Lambda_S_MotorSingle; % Indep damage to Motor #1
P(7) = Lambda_S_MotorSingle; % Indep damage to Motor #2
P(8) = Lambda_S_ValveSingle; % Indep damage to Valve #1
P(9) = Lambda_S_ValveSingle; % Indep damage to Valve #2
P(10) = Lambda_S_ValveSingle;% Indep damage to Valve #3
P(11) = Lambda_S_ValveSingle;% Indep damage to Valve #4

P(12) = Lambda_S_MotorDouble; % CC damage to both motors
P(13) = Lambda_S_PumpDouble; % CC damage to pump #1 and #2
P(14) = Lambda_S_PumpDouble; % CC damage to pump #1 and #3
P(15) = Lambda_S_PumpDouble; % CC damage to pump #2 and #3
P(16) = Lambda_S_PumpTriple; % CC damage to all three pumps

P(17) = Lambda_S_ValveDouble; % CC damage to Valve #1 and#2
P(18) = Lambda_S_ValveDouble; % CC damage to Valve #1 and#3
P(19) = Lambda_S_ValveDouble; % CC damage to Valve #1 and#4
P(20) = Lambda_S_ValveDouble; % CC damage to Valve #2 and#3
P(21) = Lambda_S_ValveDouble; % CC damage to Valve #2 and#4
P(22) = Lambda_S_ValveDouble; % CC damage to Valve #3 and#4

P(23) = Lambda_S_ValveTriple; % CC damage to Valve #1,2 &3
P(24) = Lambda_S_ValveTriple; % CC damage to Valve #1,2 &4
P(25) = Lambda_S_ValveTriple; % CC damage to Valve #1,3 &4
```

```
P(26) = Lambda_S_ValveTriple; % CC damage to Valve #2,3 &4
P(27) = Lambda_S_ValveQuad; % CC damage to all Valves
P;

% Define the maximum total event rate, which must be  greater than the
% sum of the individual event rates evaluated over t_zero to t_max

if AlphaR_Tank >1,
    R_Tank_Max =
AlphaR_Tank*LambdaR_Tank^AlphaR_Tank*tmax^(AlphaR_Tank-1);
else R_Tank_Max = LambdaR_Tank;
end

if AlphaR_PumpSingle >1,
    R_PumpSingle_Max = AlphaR_PumpSingle*...
        LambdaR_PumpSingle^AlphaR_PumpSingle*tmax^(AlphaR_PumpSingle-
1);
else R_PumpSingle_Max = LambdaR_PumpSingle;
end

if AlphaR_PumpDouble >1,
    R_PumpDouble_Max = AlphaR_PumpDouble*...
        LambdaR_PumpDouble^AlphaR_PumpDouble*tmax^(AlphaR_PumpDouble-
1);
else R_PumpDouble_Max = LambdaR_PumpDouble;
end

if AlphaR_PumpTriple >1,
    R_PumpTriple_Max = AlphaR_PumpTriple*...
        LambdaR_PumpTriple^AlphaR_PumpTriple*tmax^(AlphaR_PumpTriple-
1);
else R_PumpTriple_Max = LambdaR_PumpTriple;
end

if AlphaR_Turbine >1,
    R_Turbine_Max = AlphaR_Turbine*...
        LambdaR_Turbine^AlphaR_Turbine*tmax^(AlphaR_Turbine-1);
else R_Turbine_Max = LambdaR_Turbine;
end

if AlphaR_MotorSingle >1,
    R_MotorSingle_Max = AlphaR_MotorSingle*...

LambdaR_MotorSingle^AlphaR_MotorSingle*tmax^(AlphaR_MotorSingle-1);
else R_MotorSingle_Max = LambdaR_MotorSingle;
end

if AlphaR_MotorDouble >1,
    R_MotorDouble_Max = AlphaR_MotorDouble*...

LambdaR_MotorDouble^AlphaR_MotorDouble*tmax^(AlphaR_MotorDouble-1);
else R_MotorDouble_Max = LambdaR_MotorDouble;
end

if AlphaR_ValveSingle >1,
```

```
    R_ValveSingle_Max = AlphaR_ValveSingle*...
LambdaR_ValveSingle^AlphaR_ValveSingle*tmax^(AlphaR_ValveSingle-1);
else R_ValveSingle_Max = LambdaR_ValveSingle;
end

if AlphaR_ValveDouble >1,
    R_ValveDouble_Max = AlphaR_ValveDouble*...
LambdaR_ValveDouble^AlphaR_ValveDouble*tmax^(AlphaR_ValveDouble-1);
else R_ValveDouble_Max = LambdaR_ValveDouble;
end

if AlphaR_ValveTriple >1,
    R_ValveTriple_Max = AlphaR_ValveTriple*...
LambdaR_ValveTriple^AlphaR_ValveTriple*tmax^(AlphaR_ValveTriple-1);
else R_ValveTriple_Max = LambdaR_ValveTriple;
end

if AlphaR_ValveQuad >1,
    R_ValveQuad_Max = AlphaR_ValveQuad*...
        LambdaR_ValveQuad^AlphaR_ValveQuad*tmax^(AlphaR_ValveQuad-1);
else R_ValveQuad_Max = LambdaR_ValveQuad;
end

MER_Part1 = R_Tank_Max + 3*R_PumpSingle_Max + 3*R_PumpDouble_Max...
    + R_PumpTriple_Max;

MER_Part2 = R_Turbine_Max + 2*R_MotorSingle_Max + R_MotorDouble_Max;
MER_Part3 = 4*R_ValveSingle_Max + 6*R_ValveDouble_Max...
    + 4*R_ValveTriple_Max + R_ValveQuad_Max;

MER = 2*(MER_Part1 + MER_Part2 + MER_Part3); % 2X safety factor on MER

event_counter = 0;

% START THE SIMULATION
for k =1:n
    X = InitialX; % Initialize system state vector
    t = 0; %Reset system clock

    % Generate damage or failure on demand:
    if Rand_Tankstart == 1, Tankstart = rand; else Tankstart =0; end
    if Rand_Pumpstart == 1, Pumpstart = rand; else Pumpstart =0; end
    if Rand_Pumpstart2 == 1, Pumpstart2 = rand; else Pumpstart2 =0; end
    if Rand_Pumpstart3 == 1, Pumpstart3 = rand; else Pumpstart3 =0; end
    if Rand_Turbinestart == 1, Turbinestart = rand; else
Turbinestart=0; end
    if Rand_Motorstart == 1, Motorstart = rand; else Motorstart =0; end
    if Rand_Motorstart2 == 1, Motorstart2 = rand; else Motorstart2 =0;
end
    if Rand_Valvestart == 1, Valvestart = rand; else Valvestart =0; end
    if Rand_Valvestart2 == 1, Valvestart2 = rand; else Valvestart2 =0;
end
```

```
    if Rand_Valvestart3 == 1, Valvestart3 = rand; else Valvestart3 =0;
end
    if Rand_Valvestart4 == 1, Valvestart4 = rand; else Valvestart4 =0;
end

 % Populate the event matrix

    E(1,1) = Tankstart; % damage to Tank or inlet piping
    E(2,2) = Pumpstart; % damage to Pump #1
    E(3,3) = Pumpstart; % damage to Pump #2
    E(4,4) = Pumpstart; % damage to Pump #3
    E(5,5) = Turbinestart; % damage to Turbine
    E(6,6) = Motorstart; % damage to Motor #1
    E(7,7) = Motorstart; % damage to Motor #2
    E(8,8) = Valvestart; % damage to Valve #1
    E(9,9) = Valvestart; % damage to Valve #2
    E(10,10) = Valvestart; % damage to Valve #3
    E(11,11) = Valvestart; % damage to Valve #4

    % Common cause events for pump group
    E(12,2:3) = Pumpstart2; % CC damage to Pump #1 and #2
    E(13,2) = Pumpstart2; % CC damage to Pump #1 and #3
    E(13,4) = Pumpstart2; % continued
    E(14,3:4) = Pumpstart2; % CC damage to Pump #2 and #3
    E(15,2:4) = Pumpstart3; % CC damage to all three pumps

    % Common cause events for motor group
    E(16,6:7) = Motorstart2; % CC damage to both motors

    % Common cause events for valve group
    E(17,8:9) = Valvestart2; % CC damage to valve 1 and 2
    E(18,8) = Valvestart2; % CC damage to valve 1 and 3
    E(18,10) = Valvestart2; % continued
    E(19,8) = Valvestart2; % CC damage to valve 1 and 4
    E(19,11) = Valvestart2; % continued
    E(20,9:10) = Valvestart2; % CC damage to valve 2 and 3
    E(21,9) = Valvestart2; % CC damage to valve 2 and 4
    E(21,11) = Valvestart2; % continued
    E(22,10:11) = Valvestart2; % CC damage to valve 3 and 4

    E(23,8:10) = Valvestart3; % CC damage to valve 1,2, and 3
    E(24,8:9) = Valvestart3; % CC damage to valve 1,2, and 4
    E(24,11) = Valvestart3; % continued
    E(25,8) = Valvestart3; % CC damage to valve 1, 3, and 4
    E(25,10:11) = Valvestart3; % continued
    E(26,9:11) = Valvestart3; % CC damage to valves 2,3, and 4

    E(27,8:11) = Valvestart4; % CC damage to all four valves

    % Row 28 of E is the null event
    E;

    for j = 1:27
        OV1 = zeros(1,28); % Initialize Occurrence vector
        OV1 (j) = 1;
```

```
        z1 = rand;
        if  P(j) > z1;
            EC1 = OV1*E; % Event consequence - selects a row from E
               % corresponding to the event type
            X = X .* EC1;
            for z = 1:11
                if X(z)<DTV(z);
                   X(z) =0;
                end
            end
        end
    end
    X;
    % Check if system is still functioning

    % Define terms in structure function
        PathA = X(1)*X(2)*X(6)*X(8);
        PathB = X(1)*X(3)*X(5)*X(8);
        PathC = X(1)*X(2)*X(6)*X(9);
        PathD = X(1)*X(3)*X(5)*X(9);
        PathE = X(1)*X(3)*X(5)*X(10);
        PathF = X(1)*X(4)*X(7)*X(10);
        PathG = X(1)*X(3)*X(5)*X(11);
        PathH = X(1)*X(4)*X(7)*X(11);
        % Success criteria for MD Pumps:
        Motor_vector = [PathA*PathC,PathA*PathF,PathA*PathH,...
            PathC*PathF,PathC*PathH,PathF*PathH ];
        MotorPath = max(Motor_vector);
        % Success criteria for TD Pumps:
        Turbine_vector = [PathB*PathD,PathB*PathE,PathB*PathG,...
            PathD*PathE,PathD*PathG,PathE*PathG ];
        TurbinePath = max(Turbine_vector);

        SF = max(MotorPath,TurbinePath);% Compute Structure function


    while SF > 0 && t < 1.5*tmax
        OV2 = zeros(1,28); % Initialize Occurence vector
        t = t - 1 / MER * log(rand); % Generate time to next event
        event_counter = event_counter +1;

        % FAILURE DURING RUN: DETERMINE THE RATES FOR EACH EVENT

        % Find component independent failure rate as a function of t
        % using Weibull distribution

        % Event rates
        R = zeros(1,28);
        % Single component event rates
        R (1) = AlphaR_Tank*LambdaR_Tank^AlphaR_Tank*t^(AlphaR_Tank-1);

        R_PumpSingle = AlphaR_PumpSingle*...
            LambdaR_PumpSingle^AlphaR_PumpSingle*t^(AlphaR_PumpSingle-
1);
```

```
        R (2:4) = R_PumpSingle;

        R_MotorSingle = AlphaR_MotorSingle*LambdaR_MotorSingle...
            ^AlphaR_MotorSingle*t^(AlphaR_MotorSingle-1);

        R (5) = AlphaR_Turbine*LambdaR_Turbine^AlphaR_Turbine*...
            t^(AlphaR_Turbine-1);

        R(6:7) = R_MotorSingle;

        R_ValveSingle = AlphaR_ValveSingle*LambdaR_ValveSingle...
            ^AlphaR_ValveSingle*t^(AlphaR_ValveSingle-1);

        R(8:11) = R_ValveSingle;

        % Common cause event rates

        R(12) =
AlphaR_MotorDouble*LambdaR_MotorDouble^AlphaR_MotorDouble...
            *t^(AlphaR_MotorDouble-1);

        R_PumpDouble = AlphaR_PumpDouble*LambdaR_PumpDouble...
            ^AlphaR_PumpDouble*t^(AlphaR_PumpDouble-1);


        R(13:15) = R_PumpDouble;

        R(16) =
AlphaR_PumpTriple*LambdaR_PumpTriple^AlphaR_PumpTriple...
            *t^(AlphaR_PumpTriple-1);

        R_ValveDouble = AlphaR_ValveDouble*LambdaR_ValveDouble...
            ^AlphaR_ValveDouble*t^(AlphaR_ValveDouble-1);

        R(17:22) = R_ValveDouble;

        R_ValveTriple = AlphaR_ValveTriple*LambdaR_ValveTriple...
            ^AlphaR_ValveTriple*t^(AlphaR_ValveTriple-1);

        R(23:26) = R_ValveTriple;

        R(27) = AlphaR_ValveQuad*LambdaR_ValveQuad^AlphaR_ValveQuad...
            *t^(AlphaR_ValveQuad-1);

        % Null Event Rate
        R (28) = MER - sum(R);

        % Additional check on Max Event Rate
        if R(28)<0
            fprintf 'increase MER'
            pause
        end
```

```
        % Generate damage or failure during run:
        if Rand_Tankrun == 1, Tankrun = rand; else Tankrun=0;end
        if Rand_Pumprun == 1, Pumprun = rand; else Pumprun=0; end
        if Rand_Pumprun2 == 1, Pumprun2 = rand; else Pumprun2=0; end
        if Rand_Pumprun3 == 1, Pumprun3 = rand; else Pumprun3=0; end
        if Rand_Turbinerun == 1, Turbinerun = rand; else Turbinerun=0;
end
        if Rand_Motorrun == 1, Motorrun = rand; else Motorrun=0; end
        if Rand_Motorrun2 == 1, Motorrun2 = rand; else Motorrun2=0; end
        if Rand_Valverun == 1, Valverun = rand; else Valverun=0; end
        if Rand_Valverun2 == 1, Valverun2 = rand; else Valverun2=0; end
        if Rand_Valverun3 == 1, Valverun3 = rand; else Valverun3=0; end
        if Rand_Valverun4 == 1, Valverun4 = rand; else Valverun4=0; end

        % Populate Event Matrix
        E(1,1) = Tankrun; % damage to Tank or inlet piping
        E(2,2) = Pumprun; % damage to Pump #1
        E(3,3) = Pumprun; % damage to Pump #2
        E(4,4) = Pumprun; % damage to Pump #3
        E(5,5) = Turbinerun; % damage to Turbine
        E(6,6) = Motorrun; % damage to Motor #1
        E(7,7) = Motorrun; % damage to Motor #2
        E(8,8) = Valverun; % damage to Valve #1
        E(9,9) = Valverun; % damage to Valve #2
        E(10,10) = Valverun; % damage to Valve #3
        E(11,11) = Valverun; % damage to Valve #4

        % Common cause events for pump group
        E(12,2:3) = Pumprun2; % CC damage to Pump #1 and #2
        E(13,2) = Pumprun2; % CC damage to Pump #1 and #3
        E(13,4) = Pumprun2; % continued
        E(14,3:4) = Pumprun2; % CC damage to Pump #2 and #3
        E(15,2:4) = Pumprun3; % CC damage to all three pumps

        % Common cause events for motor group
        E(16,6:7) = Motorrun2; % CC damage to both motors

        % Common cause events for valve group
        E(17,8:9) = Valverun2; % CC damage to valve 1 and 2
        E(18,8) = Valverun2; % CC damage to valve 1 and 3
        E(18,10) = Valverun2; % continued
        E(19,8) = Valverun2; % CC damage to valve 1 and 4
        E(19,11) = Valverun2; % continued
        E(20,9:10) = Valverun2; % CC damage to valve 2 and 3
        E(21,9) = Valverun2; % CC damage to valve 2 and 4
        E(21,11) = Valverun2; % continued
        E(22,10:11) = Valverun2; % CC damage to valve 3 and 4

        E(23,8:10) = Valverun3; % CC damage to valve 1,2, and 3
        E(24,8:9) = Valverun3; % CC damage to valve 1,2, and 4
        E(24,11) = Valverun3; % continued
        E(25,8) = Valverun3; % CC damage to valve 1, 3, and 4
        E(25,10:11) = Valverun3; % continued
        E(26,9:11) = Valverun3; % CC damage to valves 2,3, and 4

        E(27,8:11) = Valverun4; % CC damage to all four valves
```

```
        % Row 28 of E is the null event
        E;

        EP2 = R / MER; % Determine probability of each event
        z2 = rand;
        ET2 = sum(z2>cumsum(EP2))+1; % Event type
        OV2 (ET2) = 1; % Puts a one in the element of OV corresponding
                       % to the event type
        EC2 = OV2*E; % Event consequence - selects a row from E
                     % corresponding to the event type
        X = X .* EC2;
        for z = 1:11
            if X(z)<DTV(z);
               X(z) = 0;
            end
        end
        X;

        % Check if system is still functioning

        % Define terms in structure function
        PathA = X(1)*X(2)*X(6)*X(8);
        PathB = X(1)*X(3)*X(5)*X(8);
        PathC = X(1)*X(2)*X(6)*X(9);
        PathD = X(1)*X(3)*X(5)*X(9);
        PathE = X(1)*X(3)*X(5)*X(10);
        PathF = X(1)*X(4)*X(7)*X(10);
        PathG = X(1)*X(3)*X(5)*X(11);
        PathH = X(1)*X(4)*X(7)*X(11);
        % Success criteria for MD Pumps:
        Motor_vector = [PathA*PathC,PathA*PathF,PathA*PathH,...
            PathC*PathF,PathC*PathH,PathF*PathH ];
        MotorPath = max(Motor_vector);
        % Success criteria for TD Pumps:
        Turbine_vector = [PathB*PathD,PathB*PathE,PathB*PathG,...
            PathD*PathE,PathD*PathG,PathE*PathG ];
        TurbinePath = max(Turbine_vector);

        SF = max(MotorPath,TurbinePath);% Compute Structure function

    end
    timevector(k) = t;
end

% PLOT THE SIMULATION RESULTS

timesort = sort(timevector); % puts the failure times in
                             % increasing order
y_axis_counter = 1:n;

hold on
p = plot (timesort, (n-y_axis_counter+1)/n);
set(p,'Color','blue','LineWidth',1.05)
```

```
% define x and y axes; comment out the undesiredx and y limits

xlim ([0,tmax])
ylim ([0,1])
%xlim ([0,tmax])
%ylim ([0.998,1])
xlabel 'time in hours'
ylabel 'probability system is operable'

% COMPARE TO THE ANALYTIC SOLUTION
% Note that some simplifications are made (Rare event, etc.)
R = ones(1,tmax);
R2 = ones(1,tmax);
for z = 1:tmax
    V1 = Lambda_S_ValveSingle + LambdaR_ValveSingle*z;
    V2 = Lambda_S_ValveDouble + LambdaR_ValveDouble*z;
    V3 = Lambda_S_ValveTriple + LambdaR_ValveTriple*z;
    V4 = Lambda_S_ValveQuad + LambdaR_ValveQuad*z;
    P1 = Lambda_S_PumpSingle + LambdaR_PumpSingle*z;
    P2 = Lambda_S_PumpDouble + LambdaR_PumpDouble*z;
    P3 = Lambda_S_PumpTriple + LambdaR_PumpTriple*z;
    M1 = Lambda_S_MotorSingle + LambdaR_MotorSingle*z;
    M2 = Lambda_S_MotorDouble + LambdaR_MotorDouble*z;
    T = Lambda_S_Turbine + LambdaR_Turbine*z;
    C = Lambda_S_Tank + LambdaR_Tank*z;



    Terms = C+4*V1^3+(P1+T)*(P1+M1)*(P1+M1)...
        +4*V1*(P1+T)*(P1+M1)...
        +V4+P3+4*V3+12*V2*V1+15*V2^2 ...
        +10*P1*V2*(P1+T)+10*P1*V2*M1 ...
        +10*V2*P2+10*V2*M1*T+3*P1*P2+3*P2^2 ...
        +2*P2*M1+2*P2*M2+4*P2*V1+P2*T+M2*P1+M2*T;

    R(z) = 1-Terms;

end

p = plot (R);
set(p,'Color','red','LineWidth',1.05)

toc
event_counter
```

SECTION A2 - SWARM SIMULATION

```
% SWARM_SIMULATION
clear all
clc
format compact

% INPUT MODEL PARAMETERS
x = 100; % Number of robots
X_Max =100; % Size of search area
Y_Max =100;
t_Max = 400; % Length of simulation

max_speed = 100;
min_speed = 50;

Change_Course_Rate = 20;

% SPECIFY ROBOT FAILURE RATE PARAMETERS
a = 0.001; % Constant part of failure rate
b = 0.002; % Failure rate time parameter
c = 0.0001; % Failure rate X location parameter
d = 0.0001; % Failure rate Y location paramter
Beta = -0.2; % Failure rate distance parameter
CCF_Radius = 10;

Max_Failure_Rate = x * (a + b * t_Max^(1/3) + c * X_Max^(1/3) + d *
Y_Max^(1/3));
MER = Change_Course_Rate + Max_Failure_Rate;

q = round(MER*t_Max*1.5); % upper bound on number of events


% LOOP AROUND BETA

while Beta < 0.6;
Beta = Beta + 0.2;
% NUMBER OF SIMULATION RUNS
timesortcum = zeros(1,x);
k = 100;
Total_search_time = zeros(1,k);

for j=1:k;

% INITIALIZE SYSTEM
t=0; % initialize time
t_last_course = 0;
t_last_failure = 0;
n=0; % event counter
RSV = ones (1,x); % Robot survival vector- Specifies all robots
operable
Failure_time = t_Max * ones(1,x);
```

```
% FIND INITIAL POSITIONS AND VELOCITIES
SearchBoundaries = [X_Max,Y_Max];
X_Pos = rand(1,x)*X_Max;
Y_Pos = rand(1,x)*Y_Max;

Speed = rand (1,x)*(max_speed - min_speed)+ones(1,x)*min_speed;
Direction = rand(1,x)*2*pi;
X_Velocity = Speed .* cos(Direction);
Y_Velocity = Speed .* sin(Direction);

% START SIMULATION
while t<t_Max
% Event counter
n = n+1;

% GENERATE TIME TO EVENT
t= t - 1 / MER * log(rand);

time_vector(n) = t;

% EVALUATE EVENT RATES AT THAT TIME

FRV = ones (1,x) .* (a + b*t^(1/3) + c .* (X_Pos .^ (1/3)) + d .*
(Y_Pos .^ (1/3)));
FRV = FRV .* RSV;
% Generates independent failure rates for surviving robots

TFR = sum (FRV); % Total failure rate
Null = MER - (TFR + Change_Course_Rate);

% DETERMINE TYPE OF EVENT

Long_array = [FRV Change_Course_Rate Null];
EP2 = Long_array / MER; % Determine probability of each event
event_type = sum(rand>cumsum(EP2))+1; % Event type


% FOR COURSE CHANGE EVENTS DETERMINE POSITION AND NEW COURSE

while event_type == x+1

Speed = rand (1,x)*(max_speed - min_speed)+ones(1,x)*min_speed;
Direction = rand(1,x)*2*pi;
X_Velocity = Speed .* cos(Direction);
Y_Velocity = Speed .* sin(Direction);

X_Pos = X_Pos + X_Velocity *(t-t_last_course);
A = X_Pos < X_Max;
B = A - ones(1,x);
X_Pos = X_Pos .*A - X_Max * B;
C = X_Pos > 0;
X_Pos = X_Pos .*C;
```

```
Y_Pos = Y_Pos + Y_Velocity*(t-t_last_course);
A = Y_Pos < Y_Max;
B = A - ones(1,x);
Y_Pos = Y_Pos .*A - Y_Max * B;
C = Y_Pos > 0;
Y_Pos = Y_Pos .*C;


event_type = x+2;
t_last_course = t;

end

% FOR FAILURE EVENTS DETERMINE CONSEQUENCES OF FAILURE

while event_type < (x + 1)
RSV(event_type) = 0;
Failure_time(event_type) = t;

% Robot positions
X_Pos = X_Pos + X_Velocity *(t-t_last_course);
A = X_Pos < X_Max;
B = A - ones(1,x);
X_Pos = X_Pos .*A - X_Max * B;
C = X_Pos > 0;
X_Pos = X_Pos .*C;

Y_Pos = Y_Pos + Y_Velocity*(t-t_last_course);
A = Y_Pos < Y_Max;
B = A - ones(1,x);
Y_Pos = Y_Pos .*A - Y_Max * B;
C = Y_Pos > 0;
Y_Pos = Y_Pos .*C;

% Pick a Robot
Robot_number = event_type;

Robot_X_location = X_Pos(Robot_number)*ones(1,x);
Robot_Y_location = Y_Pos(Robot_number)*ones(1,x);

% Find distance to other robots

X_Distance = X_Pos - Robot_X_location;
Y_Distance = Y_Pos - Robot_Y_location;
Distance = (X_Distance.^2 + Y_Distance.^2).^0.5;

% Evaluate failures of other robots

CCFRV = CCF_Radius * ones(1,x);
Veh_outside_zone = Distance > CCFRV;
Veh_inside_zone = Distance < CCFRV;
Veh_CCF_Fail = Beta * Veh_inside_zone > rand(1,x);
Veh_CCF_Survive = ones(1,x) - Veh_CCF_Fail;
```

```
New_failures = RSV - RSV .*Veh_CCF_Survive;
RSV = RSV .* Veh_CCF_Survive;

Failure_time = Failure_time - (t_Max - t)*New_failures;

event_type = x+2;
end

RSV_Total(n) = sum (RSV);
end

% AVERAGE ALL OF THE DATA? - need to average over k

timesort = sort(Failure_time); % puts the failure times in increasing
order
timesortcum = timesortcum + sort(Failure_time);
Total_search_time (j) = sum (Failure_time);

end
%scatter (X_Pos, Y_Pos) $scatter plot of robot positions
timesortave = (1/k) * timesortcum;

y_axis_counter = 1:x;
hold on
p = plot (timesortave, (x-y_axis_counter+1));
set(p,'Color','blue','LineWidth',1.05)

% define x and y axes; comment out the undesired x and y limits

xlim ([0,t_Max])
ylim ([0,x])
%xlim ([0,tmax])
%ylim ([0.998,1])
xlabel 'time'
ylabel 'number of vehicles'

end
```