Customer–Centered Reliability Measures For Flexible Multistate Reliability Models

by

Russell Brunelle

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Doctor of Philosophy

University of Washington

1998

Approved by_____

(Chairperson of Supervisory Committee)

_____

_____

_____

_____

Program Authorized
to Offer Degree_____

Date_____

In presenting this dissertation in partial fulfillment of the requirements for the Doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of this dissertation is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to University Microfilms, 1490 Eisenhower Place, P.O. Box 975, Ann Arbor, MI 48106, to whom the author has granted "the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform."

Signature_____

Date_____

University of Washington

Abstract

Customer–Centered Reliability Measures For Flexible Multistate Reliability Models

by Russell Brunelle

Chairperson of Supervisory Committee
Dr. Kailash C. Kapur
Industrial Engineering Program

Most models and tools for analyzing reliability have traditionally been based on binary approximations; these models and tools are expanded in this dissertation to allow multistate and continuum analysis. A series of customer-centered reliability measures, valid for binary, multistate, and continuum reliability models, is defined and developed. Techniques are presented for computing or approximating the state of the system as a function of the states of its components, based on subsystem identification and an application of multiquadric interpolation, and an extension of the interpolation method is presented which will preserve any monotonicity in the original customer-supplied data set. Methods for multistate reliability modeling based on continuous-time Markov chains are developed and extended. Procedures for assessing the accuracy of a continuum model discretization are proposed, and reliability measure computations for non-trivial continuum and mixed systems are illustrated. A comprehensive software package which performs non-binary reliability analyses is included and documented.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Dr. Kailash C. Kapur for his assistance and guidance; this dissertation would not have been possible without the benefit of his far-reaching and deep insights into reliability. In addition, thanks are due to Dr. Stephen Wolfram, who provided an inspiring hint at a critical phase in this project.

Chapter 1

INTRODUCTION

1.1 Motivation

The approach which has traditionally been used for reliability modeling is to take one of a product or service's quality characteristics, functionality, discretize it into a binary variable, and then use either "mean time to failure" or "the probability it is still functioning" to quantify its reliability. Clearly, there *are* products and services which the customer may honestly experience only as either functioning or failing to function,[1] and there *are* products and services for which the customer might identify either "mean time to failure" or "the probability it is still functioning" as actually being synonymous with customer satisfaction. However, when the customer's experience or desires differ from this alternatives should be available.

The majority of this dissertation's efforts stem from precisely two assumptions:

1. That a customer's experience of product degradation over time may often be more refined than only noticing "perfect functionality" and "complete failure."

2. That a customer's assessment of a product's reliability can sometimes depend on more than the product's *current* state of functioning — that the product's performance at earlier times may also have an impact.

Stated plainly, the goal of this dissertation is to provide reliability practitioners

[1] such as certain elementary machine parts

and theorists with new models and tools based on these two assumptions. By expanding the selection of available reliability measures along with the range of states which products can be modeled to assume, numerical assessments of reliability may become more capable of approximating what the customer's assessment of the product's reliability will ultimately be. If this goal is achieved, it may become easier for other researchers and engineers to incorporate reliability into more general models of quality and customer satisfaction. Given the degree to which modern engineering and management are re-orienting themselves to become more "customer-centered," this approach may prove to be of real-world value. The approach this dissertation takes is also consistent with what at least one reliability researcher [250] has recommended as potential goals for reliability engineering (see Table 1.1).

Table 1.1: Reliability Engineering Goals

TRADITIONAL	IDEAL
Measurement (Statistical Estimation)	Improvement
Binary	Multistate, Continuous
Accept times as a noise	Achieve robustness
Probability models	Utility, Customer satisfaction, Basis for action
Statistical studies (Enumerative)	Time — future (Analytical study)
Failure	Transition, One state to another
System Models (Structure function)	Customer Driven System Models

1.2 What Other Researchers Have Done

A large number of papers and books (most notably, [21, 66, 56]) have been written on binary reliability models, which allow only two states of functioning for a system and

for each of its components (perfect functionality and complete failure). In recognition of the fact that many systems exhibit noticeable gradations of performance between these two extremes, over 100 papers (notably, [121, 115, 151, 163, 146]) have been written in the last 20 years on the multistate model, which allows for any finite number of possible states. In the last 14 years, approximately 10 papers (essentially, [198, 195, 196, 197, 205, 207] and a few others) have been written on the continuum model, which allows for a continuum of possible states.

In the multistate model literature (see Appendix A), a large amount of attention has been devoted to questions of "coherence": how "unreasonable" systems may be eliminated from consideration so that more powerful theorems may be proved about those systems which were not eliminated. It is not clear how this benefits the customer in any meaningful way.

With the exception of a few recent papers for multistate models [192, 145, 188, 194, 189], the question of what reliability measures to use for non-binary reliability models has been completely unexplored. No papers whatsoever have addressed the question of what reliability measures to use with continuum systems.

1.3 This Dissertation's Original Contributions

For all model types, a major contribution of this dissertation is the creation and development of a wide variety of reliability measures (Chapter 2). In some cases these are new measures created during the course of this research, and in other cases they are measures which had not yet been applied to the continuum case. Nowhere else in the literature are reliability measures for binary, multistate, and continuum models considered together with a common, unified notation.

The question of identifying the relationship between the system's state and the states of its components is addressed by drawing attention to the fact that specific subsets of possible structures are quite common in the reliability literature, and that

for many of these structures parameters may be systematically found based on limited customer input. Additionally, attention is drawn to the fact that systems of almost arbitrary complexity may be assessed and bounded, regardless of the complexity or nature of the components, if the complete system consists of modules from this subset (Chapter 3).

Several new contributions are made in the area of continuum reliability models. Specifically, a method for approximating the structural relationship between the components of a system and the system itself based on scattered-data interpolation is developed (Chapter 4). This method is extended to allow for the coherence of the customer-supplied data set to be insured in the resulting structure function interpolation. A new type of "partial-information structure function bound" is also proposed (Chapter 5). The use of Markov models for discrete reliability models is examined (Chapter 6), and a new method by which a given discretization may be assessed is proposed (Section 7.2).

In the continuum model literature, the examples considered are typically very small, and comments are often made to the effect that more extensive or more general computations are not practical. Another unique aspect of this dissertation is the fact that it demonstrates computational techniques, generally based on numerical integrations which take advantage of any component independence, for larger systems with complex, unusual, or mixed continuous/discrete component distributions (Chapter 8).

Minor contributions are made through the identification of a simple coherence definition which is valid for all model types (Section 1.6), in the use of mixed distributions for component states (Section 2.2), through the identification of reliability bounds which are robust to lost information (Section 7.1.3), and in the extension of parallel and series definitions to general multistate reliability models (Appendix C).

Finally, during the course of this research a complete and fully-documented software analysis package for non-binary reliability models was created (Appendices I

and J) with the hope that this would encourage others to follow and participate in non-binary reliability research. This software package is intended as an integral part of this dissertation.

1.4 Basic Notation

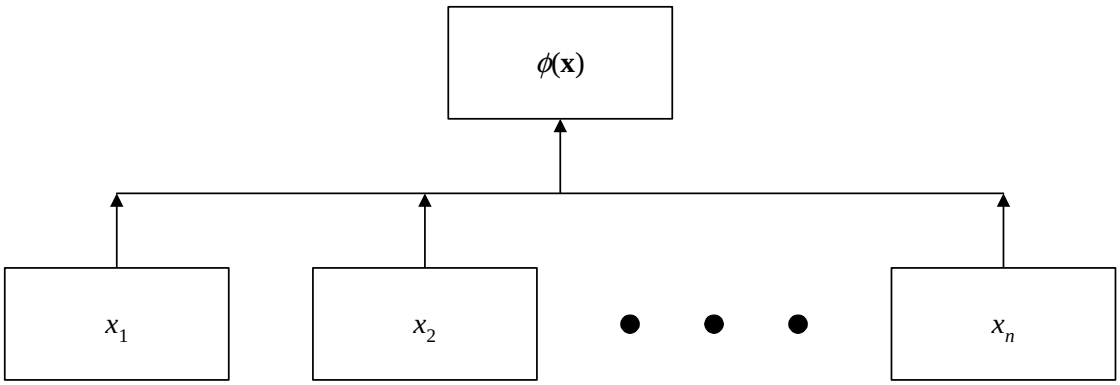


Figure 1.1: General Structure Function Diagram

The maximal state of a system or component corresponds to "perfect functioning," the minimal state corresponds to "complete failure," and other states (if any) correspond to intermediary states of functioning. We assume that, from the point of view of the customer, a state s_1 is preferable (or at least equivalent) to a state s_2 for a given system or component if $s_1 \geq s_2$.

The following notation may be utilized for any type of reliability model:

$$n \equiv \text{ Number of components constituting the system (a finite positive integer)}$$

$$C \equiv \{1, 2, \ldots, n\} \equiv \text{ The set of component indices}$$

$$\Omega_i \equiv \text{ State space of component } i,\ i \in C$$

$$M_i \equiv \max \Omega_i \equiv \text{ Best state of component } i$$

$$0 \equiv \min \Omega_i \equiv \text{ Worst state of component } i$$

$$x_i \equiv \text{ State of component } i,\ x_i \in \Omega_i$$

$$\mathbf{x} \equiv (x_1, x_2, \ldots, x_n) \equiv \text{ Component vector, } x_i \in \Omega_i$$

$$\mathbf{0} \equiv (0, 0, \ldots, 0) \in S$$

$$\mathbf{j} \equiv (j, j, \ldots, j) \in S$$

$$\mathbf{M} \equiv (M_1, M_2, \ldots, M_n)$$

$$S \equiv \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n \equiv \{\mathbf{x} \mid x_i \in \Omega_i\} \equiv \text{ Component vector space}$$

$$\phi(\mathbf{x}) \equiv \text{ Structure function yielding system state, } \phi(\mathbf{x}) \in \Omega,\ \mathbf{x} \in S$$

$$\Omega \equiv \text{ State space of the system}$$

$$M \equiv \max \Omega \equiv \text{ Best state of the system}$$

$$0 \equiv \min \Omega \equiv \text{ Worst state of the system}$$

$$S_k \equiv \{\mathbf{x} \mid \phi(\mathbf{x}) = k\} \equiv \text{ Equivalence class for system state } k$$

$$\mathbf{y} < \mathbf{x} \equiv y_i \leq x_i\ \forall i \in C, \text{ with } y_i < x_i \text{ for at least one } i \in C$$

$$\mathbf{y} \leq \mathbf{x} \equiv y_i \leq x_i\ \forall i \in C$$

$$\mathbf{y} > \mathbf{x} \equiv y_i \geq x_i\ \forall i \in C, \text{ with } y_i > x_i \text{ for at least one } i \in C$$

$$\mathbf{y} \geq \mathbf{x} \equiv y_i \geq x_i\ \forall i \in C$$

$$(k_i, \mathbf{x}) \equiv (x_1, x_2, \ldots, x_{i-1}, k, x_{i+1}, \ldots, x_n), k \in \Omega_i, i \in C, \mathbf{x} \in S$$

$$\|\mathbf{x} - \mathbf{y}\| \equiv \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

$$X_i \equiv \text{ Random variable for } x_i, \text{ the state of component } i,\ i \in C$$

$$\mathbf{X} \equiv (X_1, X_2, \ldots, X_n) \equiv \text{ Random variable of component vector}$$

$$\phi(\mathbf{X}) \equiv \text{ Structure function, yielding random variable for system state, } \phi(\mathbf{X}) \in \Omega$$

$$Q_j \equiv P[\phi(\mathbf{X}) \geq j],\ j \in \Omega$$

$$Q_{ij} \equiv P[X_i \geq j],\ j \in \Omega_i\ i \in C$$

$$\forall \equiv \text{ for all}$$

When the random variables for the states of a system's components are stochastic processes (functions of time), the following expressions (or related ones) may be used. The variable t may be dropped from expressions such as these when a time-invariant case is being considered or when time is not of interest.

$$
\begin{aligned}
t &\equiv \text{Time, } t \in [0, \infty) \\
X_i(t) &\equiv \text{The state of component } i \text{ at time } t,\ i \in C \\
\mathbf{X}(t) &\equiv (X_1(t), X_2(t), \ldots, X_n(t)) \\
F_i[s,t] &\equiv P[X_i(t) \leq s] \\
F[s,t] &\equiv P[\phi(t) \leq s] \\
\phi(t) &\equiv \phi(\mathbf{X}(t)) \equiv \text{The (random) state of the system at time } t
\end{aligned}
$$

Many examples and results in this dissertation make the assumption that the random variables representing the component states for a system, $X_1, X_2, \ldots, X_n$, are mutually independent. Discrete random variables $X_1, X_2, \ldots, X_n$ are mutually independent if and only if $P[X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n] = P[X_1 = x_1]P[X_2 = x_2] \cdots P[X_n = x_n] \,\forall\, (x_1, x_2, \ldots, x_n) \in \mathbf{R}^n$. Continuous random variables $X_1, X_2, \ldots, X_n$ are mutually independent if and only if $f[x_1, x_2, \ldots, x_n] = f_1[x_1]f_2[x_2] \cdots f_n[x_n]$ $\forall (x_1, x_2, \ldots, x_n) \in \mathbf{R}^n$, where $f_i[x_i]$ is the probability density function of X_i.

1.5 Model Types

1.5.1 Binary

The binary model allows two states of functioning for the system and for each of its components: $\phi(t) \in \{0, 1\}$, $X_i(t) \in \{0, 1\}$. These two states may be interpreted as "complete failure" and "perfect functioning." One scalar value is sufficient to characterize the stochastic behavior of a binary system or component at any moment

in time: $p(t) \equiv P[\phi(t) = 1]$, $p_i(t) \equiv P[X_i(t) = 1]$. An example of a system which exhibits binary behavior is a light bulb, which can only be on or off. An unfortunate fact is that "reliability" has often been defined in ways which imply the binary model; for example: "Reliability is the probability that a product or service will operate properly for a specified period of time under the design operating conditions without failure." [40]

The binary reliability model was the first reliability model developed, and there have been no changes in this model's definition since its formal introduction in [21] and subsequent refinement in [16]. Its early history can be summarized as follows:

- During World War I, the concept of *number of system failures per unit time* arose while comparing one, two, and four-engine military aircraft.

- During World War II, mathematician Robert Lusser derived the *product probability law of series components* while analyzing the performance of V-1 missiles.

- Around the time of the ICBM and Apollo projects, the *Binary Coherent System* was presented by Birnbaum et al. (1961), *Fault Tree Analysis* was created (1962), and *IEEE Transactions on Reliability* was founded (1963).

1.5.2 Multistate

Early papers in multistate reliability [121, 115] expanded the binary model's two states to the following set: $\phi(t) \in \{0, 1, \ldots, M\}$, $X_i(t) \in \{0, 1, \ldots, M\}$. Later researchers [150, 151, 146] further generalized this model to allow different numbers of states for the system and each component: $\phi(t) \in \{0, 1, \ldots, M\}$, $X_i(t) \in \{0, 1, \ldots, M_i\}$. The most flexible definition [188, 189] allows the states to be any non-negative number: $\phi(t) \in \{\phi_0, \phi_1, \ldots, \phi_m\}, \quad \phi_0 = 0, \quad \phi_j < \phi_k \ \forall \ j < k$, $X_i(t) \in \{x_{i0}, x_{i1}, \ldots, x_{im_i}\}, \quad x_{i0} = 0, \quad x_{ij} < x_{ik} \ \forall \ j < k$.

For the sake of reference, the first definition shall be termed the "traditional" multistate model, the second definition shall be termed the "general" multistate model, and the third definition shall be termed the "non-integerial" multistate model. The binary model is a special case of the traditional multistate model, the traditional multistate model is a special case of the general multistate model, and the general multistate model is a special case of the non-integerial multistate model. Thus, theorems proved for the non-integerial multistate model or the general multistate model will apply equally to the traditional multistate model or the binary model.[2]

The "non-integerial" multistate definition definition allows components and systems to assume states reflective of their value to the customer [192]. Unless the component and system states are allowed some real-world meaning (e.g. $\phi(t) = 3/4$ for 75% functionality, $\phi(t) = 1$ for 100% functionality, etc.) comparing different systems with different numbers of allowed states becomes problematic, measures such as "state variance" lose their robustness (i.e. they change even when unused states are added), etc. Also, in many cases it may fundamentally *not* be true that the jth state of one component has the same intrinsic value to the customer as the jth state of a different component.

An example of a component which exhibits multistate behavior is a computer memory board with many memory chips, each of which may either function or fail to function independently of the others [192]. A finite number of scalar values is sufficient to characterize the stochastic behavior of any multistate system or component at any moment in time: $p[\phi_j, t] \equiv P[\phi(t) = \phi_j] \; \forall \; \phi_j \in \Omega$ or $p[x_{ij}, t] \equiv P[X_i(t) = x_{ij}] \; \forall \; x_{ij} \in \Omega_i$.

This model's early history may be summarized as follows:

- The first non-binary reliability model (of any kind) was proposed in 1968 by Hirsch, Meisner, and Boll [54] to model reuse of military supplies. According to

[2]Often, proving a theorem for the non-integerial multistate model is only a matter of changing the notation in an existing proof for the general multistate model.

these authors, the idea for this model came from a certain Lt. Commander J. V. Reilly, Jr. (U.S. Navy, Special Projects Office and Supply Systems Command).

- The *traditional multistate model*, which requires the components and the system to have the same number of states, was independently presented in 1978 by two different groups of researchers: El-Neweihi et al. [121] and Barlow and Wu [115].

- The *general multistate model*, which allows different numbers of states for the components and the system, was developed in 1981 by Hudson [150].

1.5.3 Continuum

In 1984 and 1986, Baxter [195, 196] introduced the standard continuum reliability model, which allows a system or component's state to assume any value in a segment of the real number line: $\phi(t) \in [0,1]$, $X_i(t) \in [0,1]$. It should be pointed out that most real-world systems and components exhibit continuous degradation of some form, and so in the majority of cases the question should be whether it is feasible or necessary to use a continuum model, and not whether this model's use is philosophically justifiable. An example of a continuum system is an automobile tire, the performance of which may degrade continuously as the tread wears. The stochastic behavior of continuum systems and components may be specified through a Cummulative Distribution Function ($F[s,t] \equiv P[\phi(t) \leq s]$, $F_i[s,t] \equiv P[X_i(t) \leq s]$) or, for absolutely continuous distributions, a Probability Density Function ($f[s,t]$, $f_i[s,t]$).

In real-world analyses, one would expect few continuum systems to have absolutely continuous distributions for their states. For example, the response of an audio speaker may gradually deteriorate over time, but it is also subject to catastrophic or "hard" failure wherein the system transitions to its lowest possible state and remains there. For such a system there should be a non-zero probability of being in state 0, and so the distribution for its state should be mixed (continuous and discrete). Indeed,

this would seem to be the case for any non-repairable continuum system [207].

For the continuum model, there are an infinity of possible values in the interval $[0, 1]$ and hence an infinity of possible state vectors and equivalence classes. Many equivalence classes may contain an infinity of these vectors. Since it is philosophically the customer who specifies the structure function, we must have some way of determining or estimating it based on the customer's demands. In the binary and multistate cases, this can always be done using upper or lower boundary points (a special subset of all possible component vectors). In the continuum case this subset may be an infinite set (that cannot be obtained from the customer in a finite amount of time), which is the reason this dissertation proposes scattered data interpolation methods for continuum models.

The earliest continuum reliability model work by Block and Savits [198] considered the state space for the system and for each of the components to be the non-negative orthant of the real number line.[3] Work was continued by Baxter [195, 196], who examined two specific structure functions (ζ and η) and dealt with general topics of component relevance. Baxter and Kim [197] created a new continuum definition for boundary points along with enhanced bounds on the distribution of $\phi(\mathbf{x})$. Montero et al. [205] proposed another continuum definition for boundary points, improved on the reliability bounds proposed by Block and Savits [198], and developed a technique for approximating a continuum model through discretization.

The continuum model appears to have originally arisen out of mathematical interest; possible applications were proposed only after its initial description in the literature.

---

[3] Although this has not yet been explicitly commented on in the literature, there are an uncountable infinity of numbers in $[0, 1]$ and $[0, 1]^n$ just as there are in $\mathbf{R}$ and $\mathbf{R}_+^n$. Since one may, through a suitable non-decreasing function, map $[0, 1]$ into $\mathbf{R}$ and $[0, 1]^n$ into $\mathbf{R}_+^n$, the restriction of the state space for the system and components in continuum models to $[0, 1]$ and $[0, 1]^n$ may be done without loss of generality.

1.6 Model Coherence

Since the introduction of the binary coherent system [21], a great deal of attention has been devoted in the reliability literature (especially in the non-binary reliability literature) to definitions for coherence — specifying a set of properties which "reasonable" systems should have, and then proving theorems for this restricted class rather than for any general mapping from $S \rightarrow \Omega$. This dissertation proposes one unified definition for coherence which is valid for all model types:
Definition: A system is *coherent* if it exhibits these properties:

1. $\phi(\mathbf{x}) \geq \phi(\mathbf{y}) \quad \forall \mathbf{x} \geq \mathbf{y}$ **(monotonicity)**

2. $\phi(\mathbf{0}) = 0 \quad \phi(\mathbf{M}) = M$ **(proper extrema)**

3. $\sup_{\mathbf{x} \in S}[\phi((M_i)_i, \mathbf{x}) - \phi(0_i, \mathbf{x})] > 0 \quad \forall i \in C$ **(component relevance)**

For models where $\Omega = \Omega_i = [0, 1] \ \forall i \in C$, this definition reduces to Baxter's definition of Weak Coherence [196] for continuum models. For models where $\Omega = \Omega_i = \{0, 1, 2, \ldots, M\} \ \forall i \in C$, this definition reduces to Butler's definition of Weak Coherence [117] for multistate models. For models where $\Omega = \Omega_i = \{0, 1\} \ \forall i \in C$, this definition reduces to the standard definition of binary coherence given by Barlow and Proschan [16]. The "monotonicity" property in the above definition is also referred to as the "non-decreasing" property.

1.7 Reliability and Quality

New research [244] makes a strong case for Customer Satisfaction being a function of both Quality and Customer Expectations. In a manner consistent with this new research, this dissertation defines customer satisfaction as the degree to which, according to the customer, the quality of a product or service satisfies his or her needs and desires. According to the ISO 8402 International Standard [246, page 6], quality

is "the totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs." Note that this definition of quality implies that one may think of it as a set of variables representing a product or service's relevant features and characteristics (see Figures 1.2 and 1.3); these features and characteristics are functions of time, and may include such things as beauty, functionality, etc. In this framework, quality has as its domain *all* aspects of the product or service's performance and existence which can impact the customer, but sets aside any considerations external to that product or service.[4]

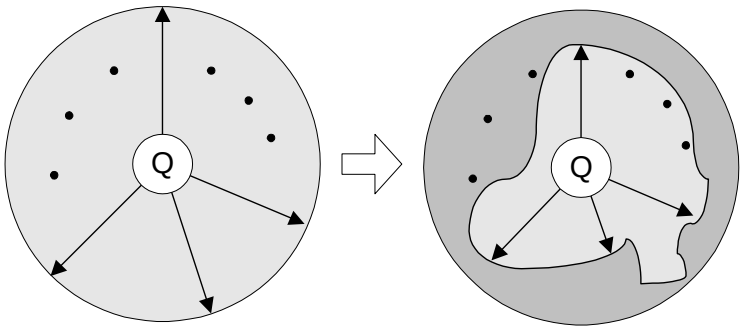


Figure 1.2: Quality Characteristic Diagram

A typical modern textbook [56, page 3] defines Reliability as "the ability of an item to perform a required function, under given environmental and operational conditions and for a stated period of time." Note that this definition is compatible with non-binary reliability models, due to the (potentially) non-binary nature of the term "ability," and the fact that "required function" may include aesthetic considerations that are important to the customer. Philosophically, the customer does not experience the reliability of an item directly; what he or she notices is changes in the quality

[4]such as responses based on changing markets, competitors' products or services, etc. which more properly fit under the "customer expectations" portion of the analysis presented in [244].

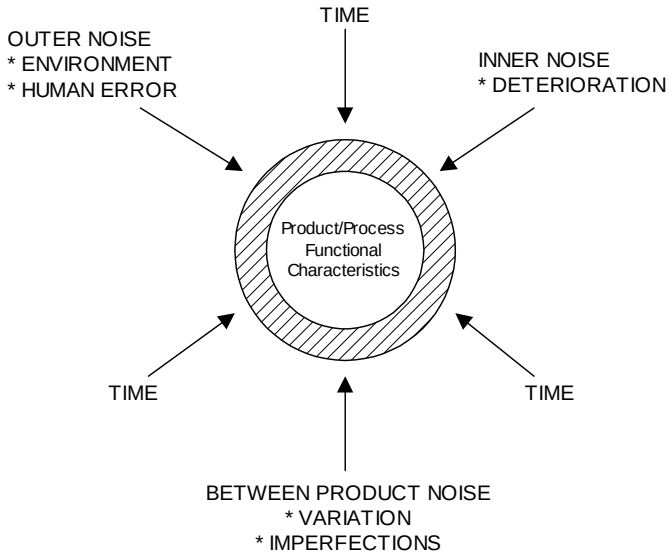


Figure 1.3: Product Noise Diagram

characteristics of that product or service, which are combined by him or her through some unknown function to affect his or her customer satisfaction.

Chapter 2

RELIABILITY MEASURES

2.1 Introduction

Because different applications and customers may require the "reliability" of a system to be strong in different ways, a number of different reliability measures should be available to the customer so that he or she may choose the one most suitable to the application at hand. As quality is defined by the customer, and reliability is a quality characteristic, the customer should be free to specify how reliability is measured for him or her. By providing a variety of different measures and suggesting situations in which each one may be a good choice, it is hoped that reliability analysts will more easily be able to find measures that match the customer's assessment of the product's performance over time.

Unfortunately, as Yang and Xue [194] point out, "Very little effort has been made in the investigation of important issues such as what the appropriate dynamic reliability measures are..." Indeed, the only reliability measures which have been mentioned for the continuum model at all so far have been the expected value $E[\phi]$ (in [195, 212]), $P[\phi \geq s]$ (in [197]), and $P[\phi > s]$ (in [198, 205, 210]). Therefore, this chapter examines a variety of customer-centered reliability measures, generalized to multistate and continuum contexts. Over 43 potential reliability measures were examined during the course of this research, and only the most valuable ones are developed in this chapter; please see the "Measures Package" in Appendix I for the additional definitions.

Table 2.1: Reliability Measure Building Blocks

	$F[s,t]$	$E[\phi(t)]$	$E[\phi^2(t)]$
Binary	$1-p(t)$ for $0 \leq s < 1$	$p(t)$	$p(t)$
Multistate	$\sum_{\forall \phi_j \leq s} p[\phi_j, t]$	$\sum_{\forall j} \phi_j p[\phi_j, t]$	$\sum_{\forall j} \phi_j^2 p[\phi_j, t]$
Cont. (PDF)	$\int_0^s f[\xi, t]\, d\xi$	$\int_0^1 s f[s,t]\, ds$	$\int_0^1 s^2 f[s,t]\, ds$
Cont. (CDF)	$F[s,t]$	$\int_0^1 (1-F[s,t])\, ds$	$\int_0^1 2s(1-F[s,t])\, ds$

2.2 Reliability Measure Computation and Definitions

Each reliability measure presented in this chapter is a function of certain basic quantities which are independent of the type of model being used (see Table 2.1). Derivations for the expressions contained in the "Cont. (CDF)" row of this table may be found in [263]. Although Table 2.1 and the reliability measures we present in this section are illustrated using system notation, applying them to components is a matter only of substituting component notation (please see Appendix E for "component importance" measures). Each of these measures is equally applicable to binary, multistate, and continuum models.

It should be noted that any multistate or continuum model may be converted to a binary model by choosing a state s_c such that any given state s is considered to be "failed" if $s \leq s_c$ and "functioning" if $s > s_c$. One defines $p(t) = 1 - F[s_c, t]$, and then may employ any of the measures and techniques developed for binary models. Measures which are constructed through this reductionist process will not be emphasized in this chapter, as they are already adequately described throughout the binary model literature. Finally, as regards the first column of Table 2.1, please note that $F[s,t] = 1\ \forall s \geq M$ and $F[s,t] = 0\ \forall s < 0$.

The functions $p(t)$, $p[\phi_j, t]$, $f[s,t]$, or $F[s,t]$, whether for the system or (more likely) for components may be available as a result of experiment, *a priori* assumptions,

or (for discrete cases) as a result of Markov modeling (see Chapter 6). For the continuum case, it is clear that continuous distributions may be required to express the probability of the system being in a particular segment of the interval $[0, 1]$ at some time t; however, discrete distributions may be required as well: for example, it is plausible that for non-repairable systems there will always be some (non-zero, increasing) probability that $P[\phi(t) = 0]$. Thus, in the most general case we will have distributions which are mixtures of discrete and continuous probability mass and density functions (the only other category of distributions, singular distributions, does not have any clear role in reliability theory [268]).

2.2.1 Moment-in-Time

These measures require knowledge of the probability distribution, first moment, or first and second moments for the system's state at only one moment in time t.

Availability

$$A[s, t] = 1 - F[s, t] = P[\phi(t) > s] \tag{2.1}$$

This measure is of value when a particular set of states is of interest to the customer (possibly states which when reached imply that the system should be repaired or discarded), or when functionality at a particular point in time (such as the end of a warranty period) is of interest. When the system under consideration is non-repairable, $A[s, \tau] \equiv R[s, \tau]$, where $R[s, \tau]$ (the "survivor function") is the probability that $\phi(t) > s \ \forall \ t \in [0, \tau]$. For non-repairable binary models, $p(t) \equiv A[0, t] = R(t) = E[\phi(t)]$. For repairable systems, one may wish to examine the steady-state availability $\mathrm{AS}[s] = \lim_{t\to\infty} A[s, t]$, which (if the limit exists) will express the steady-state probability of being above state s. For non-repairable systems, one could define $B[s, \alpha]$, which is a time such that $R[s, B[s, \alpha]] = 1 - \alpha$; for binary models $B[0, 0.10]$ would be equivalent to the traditional "B_{10} life." For

any type of system, one might be interested in examining the "average availability" $\mathrm{AV}[s,\tau] = \frac{1}{\tau}\int_0^\tau A[s,t]\,dt$, which is the average proportion of time in $[0,\tau]$ that the system will be above state s. With the exception of $B[s,\alpha]$ which has range $[0,\infty)$, these measures have range $[0,1]$ with higher values superior. A measure very similar to $A[s,t]$ was proposed by Butler for the multistate case [178], and it has also been examined in the continuum model literature.

State Expectation

$$E[\phi(t)] \tag{2.2}$$

This measure is the expected state of the system. It is intuitively familiar because it is equivalent to $R(t)$ for binary, non-repairable systems, and because it can be easily explained. It may be a useful measure in cases where (like Availability) particular times in the product's life are of interest, but (unlike Availability) no single state or states may be identified as catastrophic. For repairable systems $\mathrm{ES} = \lim_{t\to\infty} E[\phi(t)]$ may be of interest. These measures have range $[0,M]$ with higher values superior. This measure was proposed very early in the study of multistate reliability, and has also been examined in the continuum model literature [121].

State Variance

$$V[\phi(t)] = E[\phi^2(t)] - E[\phi(t)]^2 \tag{2.3}$$

There are many real-world situations in which the variance of the system state may be of great interest, such as when consistent performance of the product is critical, or when a repair schedule must be maintained without expediting. The state variance of most non-repairable systems reaches a maximum at some $t' > 0$, which may be of interest to customers concerned with consistent performance. This time may be found by solving $\frac{dV[\phi(t')]}{dt} = 0$ for t' or through global maximization. The time t'

has range $[0, \infty)$, and $V[\phi(t)]$ has range $[0, \frac{M^2}{4}]$ with lower values superior. Although the customer may not have advanced knowledge of probability, it could be easily explained that this measure provides a sense of the "spread" in the possibilities for the product's performance at a given time.

It may also be of value to consider the integral of State Variance from time 0 to time τ, called CSSD$[\tau]$.

2.2.2 Finite-Time

These measures require knowledge of the probability distribution, first moment, or first and second moments for the system's state over a span of time $t \in [0, \tau]$.

Output Expectation

$$\mathrm{OE}[\tau] = \int_0^\tau E[\phi(t)]\, dt \tag{2.4}$$

For products which are used over a period of time, a reliability measure which summarizes the product's time-dynamic performance is desirable. If a product will be used over $t \in [0, 3]$ and one design tends to degrade near $t = 1$ while another design degrades closer to $t = 2$, OE[3] would capture this distinction while $R[s, 3]$ would not, hence accurately reflecting the fact that it may be value to have use of the product's higher performance over $t \in [1, 2]$.

The measure (2.4) is the expected value of $\int_0^\tau \phi(t)\, dt$ and has range $[0, M\tau]$ (higher values superior). If one may interpret the state of the system $\phi(t)$ as being akin to "Output per Unit Time" (as one might for many process industries), then one may interpret this measure as the expected total output that will be produced over $t \in [0, \tau]$; it is the clarity of this measure's interpretation for process industries which has led to the measures (2.4), (2.5), and (2.7) being termed "output" measures in this chapter. These reliability measures are referred to as "throughput" measures

in [188], "accumulated reward" measures in [192] and "expected performance utility" measures in [194].

Many variants of this "output expectation" measure are possible. One may subtract it from $M\tau$ to obtain the Lost Output Expectation (LOE[τ]), one may divide it by $M\tau$ to obtain the Scaled Output Expectation (SOE[τ]), and one may subtract it from $M\tau$ and then divide the difference by $M\tau$ to obtain the Scaled Lost Output Expectation (SLOE[τ]).

Output Variance (Upper Bound)

Using principles described in [188, 275], we may calculate an upper bound on the variance of $\int_0^\tau \phi(t)\,dt$. This measure has range $[0, \frac{(M\tau)^2}{4}]$, lower values superior. It may be of special interest when $\int_0^\tau \phi(t)\,dt$ has a direct economic or productivity interpretation.

$$\text{OVUB}[\tau] = \left(\int_0^\tau \sqrt{E[\phi^2(t)]}\,dt\right)^2 - \left(\int_0^\tau E[\phi(t)]\,dt\right)^2 \tag{2.5}$$

One may use this measure along with Chebychev's theorem to form probability bounds around $\int_0^\tau \phi(t)\,dt$:

$$P\left[\text{OE}[\tau] - \sqrt{\frac{\text{OVUB}[\tau]}{\alpha}} < \int_0^\tau \phi(t)\,dt < \text{OE}[\tau] + \sqrt{\frac{\text{OVUB}[\tau]}{\alpha}}\right] \geq 1 - \alpha$$

2.2.3 Infinite-Time

These measures require knowledge of the probability distribution, first moment, or first and second moments for the system's state over all times $t \in [0, \infty)$.

Dwell Time Expectation

$$\text{DTE}[s] = \int_0^\infty R[s,t]\,dt \tag{2.6}$$

For non-repairable systems, DTE[s] is the expected length of time the system will remain ("dwell") above state s. In the binary case, DTE[0] $\equiv$ MTTF. Several variants of this measure are possible, including "On-Stream-Availability," which is the expected *proportion* of the system's lifetime that it spends above state s. One could similarly examine Dwell Time Variance: $\mathrm{DTV}[s] = \int_0^\infty 2tR[s,t]\,dt - \left(\int_0^\infty R[s,t]\,dt\right)^2$. DTE[$s$] may be a useful measure when the customer will begin searching for a replacement for the product as it begins to decay, though this search progress may be accelerated by further decay.

Total Output Expectation

$$\mathrm{TOE} = \int_0^\infty E[\phi(t)]\,dt \tag{2.7}$$

If the above integral converges, it may be interpreted as the average total "output" that will be delivered by the system: $\mathrm{TOE} = \lim_{\tau\to\infty} \mathrm{OE}[\tau]$. For repairable systems, $\mathrm{OESS} = \lim_{\tau\to\infty} \frac{\mathrm{OE}[\tau]}{\tau}$ may be a useful measure. For non-repairable systems one might wish to examine $T[\alpha]$, which we define as $\alpha = \frac{\mathrm{OE}[T[\alpha]]}{\mathrm{TOE}}$. Thus, $T[0.95]$ is the time at which the customer would expect to have received 95% of the total output ("benefit") the system is expected to yield.

Lifetime-Weighted Expectation

$$\mathrm{LWE}[U] = \int_0^\infty U(t)E[\phi(t)]\,dt \tag{2.8}$$

This measure attempts to capture the degree to which a product satisfies customer needs and desires over its entire lifetime. The need for such a measure was first articulated in 1969 in a paper for binary reliability models [190]. Essentially, LWE considers the performance of the product over all time, weighted by a non-negative function $U(t)$ which expresses the degree to which the customer is potentially impacted by the product's performance at each moment in time. The motivation here

is that, for many types of consumer products, it is unlikely a customer would react as strongly to product degradation late in a product's life (when it may be rarely used or may be about to be replaced with a more aesthetically or technologically advanced product) as early in the product's life. The default assumption implicit behind many binary reliability analyses — that the product is used uniformly and/or "cared about" equally until some future time when it is instantly discarded — is rarely realistic for consumer products.

$U(t)$ may also be a PDF for the time-to-occurrence of some future event; in this case, LWE may be interpreted as the expected state of the system at the time of that event. LWE reduces to TOE when one selects $U(t) = 1$, and to OE[τ] when one selects

$$U(t) = \begin{cases} 1, & 0 \leq t \leq \tau \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

Note that if $\int_0^\infty U(t)\,dt$ converges, LWE will also converge. Scaling $U(t)$ so that $\int_0^\infty U(t)\,dt = 1$ may simplify the comparison of different systems. When scaled in this way, LWE[U] has range $[0, M]$, higher values superior.

"Instant Relevance and Obsolescence" (see Figure 2.1) is well-characterized by $U(t)$ functions with shapes similar to a Uniform distribution's PDF. This weight function may be appropriate when it is known with certainty over what time interval the product is needed, and that while the product is needed it is of equal importance at all times. For example, if a new computer printer is purchased with the intent of upgrading in exactly six months, this weight function (uniform, $min = 0$ and $max = 6$) may accurately reflect the customer's level of interest in the product. Setting $min \neq 0$ may reflect buying and using the product before it is actually needed. If the product may be sold for some salvage value after time max, one could imagine utilizing the delta function at $t = max$ as part of the $U(t)$ function. Of course, if there is a time-varying pattern of customer interest in the product from min to max,

one may use some other continuously-varying PDF, truncate it at *min* and *max*, and scale.

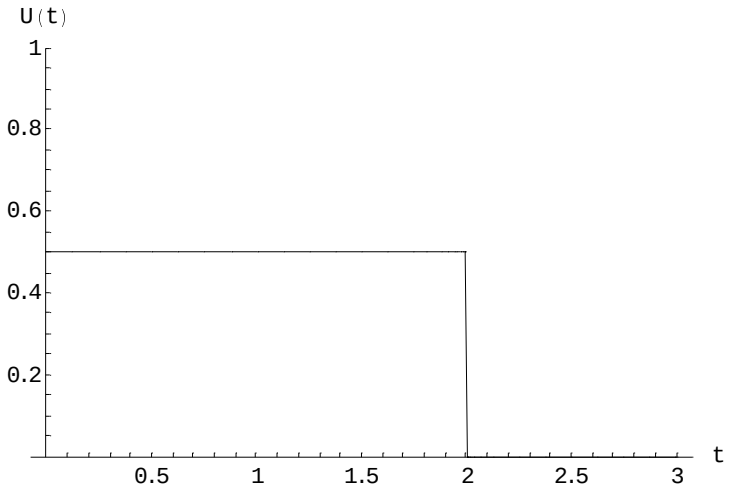


Figure 2.1: Instant Relevance and Obsolescence Example

"Gradual Obsolescence" (see Figure 2.2) is well-characterized by $U(t)$ functions with shapes similar to the Exponential PDF, the HalfNormal PDF, and the Log-Normal PDF. This class of weight functions may be appropriate in cases where the product is of greatest importance to the customer when first purchased, but becomes less important to the customer as time goes on. It may be ideally suited for industries such as the children's toys and fashion apparel industries (which exhibit inherent customer restlessness), and for consumer electronics which the consumer intends to eventually replace with more advanced technologies.

"Gradual Relevance and Obsolescence" (see Figure 2.3) is well-characterized by $U(t)$ functions with shapes similar to the Chi PDF, the ChiSquare PDF, the Gamma PDF, the Rayleigh PDF, and the Weibull PDF. This class of weight functions is of value when the product rises in importance to the customer after it is first purchased, and then later decays in importance. An example might be a product which was purchased to be used during an emergency which will occur at some unknown point

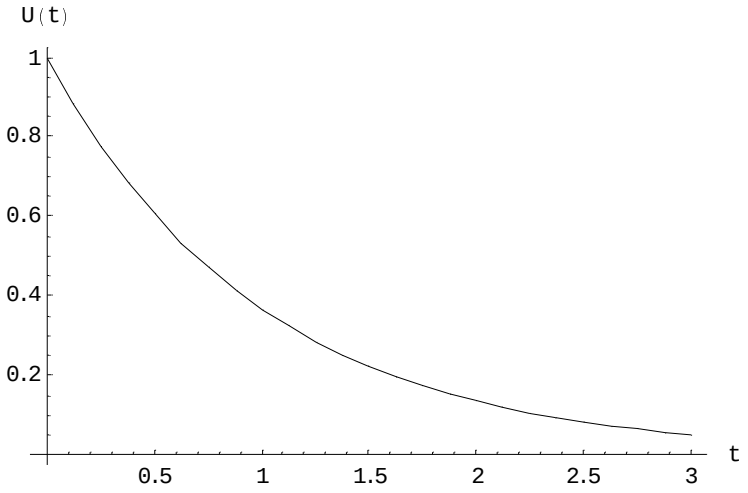


Figure 2.2: Gradual Obsolescence Example

in the future.

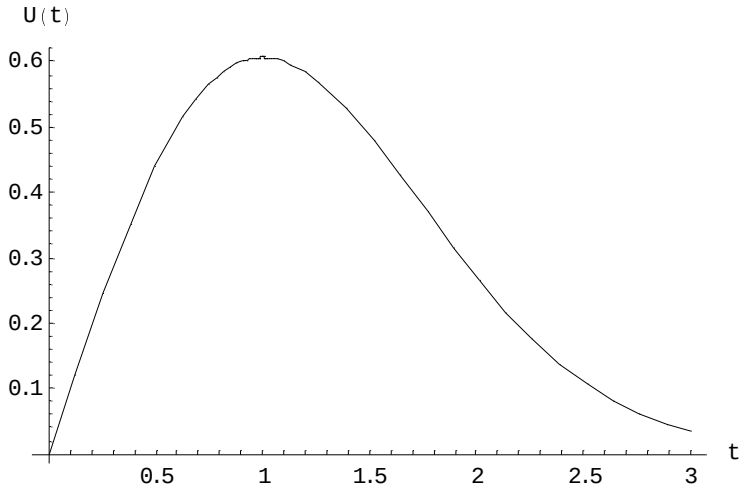


Figure 2.3: Gradual Relevance and Obsolescence Example

The only caution is that one should compute $U(t)$ in such a way that declining $U(t)$ values are not caused by declining performance of the products used to calculate it; the goal is to capture patterns of *voluntary* product-usage changes, generally due to technological obsolescence or inherent customer restlessness rather than poor

system performance in an absolute sense. Doing otherwise would neglect (by failing to reward) the possibility of improving the customer's experience of the product by exceeding his or her expectations.

2.3 Multistate Example

For this simple multistate example we assume the system begins in state $\phi_3 = 1$, transitions to state $\phi_2 = \frac{2}{3}$, then transitions to state $\phi_1 = \frac{1}{3}$, and finally transitions to state $\phi_0 = 0$ where it remains. It spends a length of time in each of the three states ϕ_3, ϕ_2, and ϕ_1 governed by an exponential distribution with common parameter λ.

In cases such as this where one has a finite number of states and exponential transition rates, continuous-time Markov chain theory may be applied to calculate the probability of being in each state as a function of time (see Chapter 6). We summarize these probabilities for this example in Table 2.2.

Table 2.2: $p[\phi_j, t]$ Values

j	ϕ_j	$p[\phi_j, t]$
0	0	$\frac{1}{2}e^{-t\lambda}(-2 + 2e^{t\lambda} - 2t\lambda - t^2\lambda^2)$
1	$\frac{1}{3}$	$\frac{1}{2}e^{-t\lambda}t^2\lambda^2$
2	$\frac{2}{3}$	$e^{-t\lambda}t\lambda$
3	1	$e^{-t\lambda}$

For this system, we can calculate $R[\phi_j, t]$ and $\mathrm{DTE}[\phi_j]$ (as given in Table 2.3 and Table 2.4) along with the selection of reliability measures given in (2.10)-(2.13).

$$E[\phi(t)] = \frac{1}{6}e^{-\lambda t}(6 + 4\lambda t + \lambda^2 t^2) \tag{2.10}$$

Table 2.3: $R[\phi_j, t]$ Values

j	ϕ_j	$R[\phi_j, t]$
0	0	$\frac{1}{2}e^{-t\lambda}(2 + 2t\lambda + t^2\lambda^2)$
1	$\frac{1}{3}$	$e^{-t\lambda}(1 + t\lambda)$
2	$\frac{2}{3}$	$e^{-t\lambda}$
3	1	0

Table 2.4: DTE$[\phi_j]$ Values

j	ϕ_j	DTE$[\phi_j]$
0	0	$\frac{3}{\lambda}$
1	$\frac{1}{3}$	$\frac{2}{\lambda}$
2	$\frac{2}{3}$	$\frac{1}{\lambda}$
3	1	0

$$V[\phi(t)] = \frac{1}{36}e^{-2\lambda t}[36(e^{\lambda t} - 1) + 16(e^{\lambda t} - 3)\lambda t + 2(e^{\lambda t} - 14)\lambda^2 t^2 - 8\lambda^3 t^3 - \lambda^4 t^4] \tag{2.11}$$

$$\text{OE}[\tau] = \frac{12 - e^{-\lambda\tau}(12 + 6\lambda\tau + \lambda^2\tau^2)}{6\lambda} \tag{2.12}$$

$$\text{TOE} = \frac{2}{\lambda} \tag{2.13}$$

Let us assume that customer usage patterns for this type of product are being assumed through comparison to similar products, and it has been found that product usage decreases exponentially over time: $U(t) = \nu e^{-t\nu}$. With this usage function, we calculate LWE as a function of λ and ν: $\text{LWE}[U] = \frac{\nu(6\lambda^2 + 8\lambda\nu + 3\nu^2)}{3(\lambda+\nu)^3}$.

For the sake of illustration, several of these measures are graphed here with $\lambda = 3$.

In Fig. 2.4 we can see the expected state of the system decreasing to 0 as t increases, as one would expect for a non-repairable system. In Fig. 2.5 we can see the variance of the system state reach its maximum at $t' = 0.60$ and then decay toward zero as the expected state of the system approaches 0. In Fig. 2.6 we can see $\mathrm{OE}[\tau]$ approach $\mathrm{TOE} = \frac{2}{3}$ as $\tau \to \infty$. In Fig. 2.7 we can see how LWE changes as a function of ν; as ν is increased, $U(t)$ decreases more rapidly, which causes the LWE reliability measure to increase towards its maximum of 1 due to the fact that the system's performance at large values of t does not impact the customer as significantly.

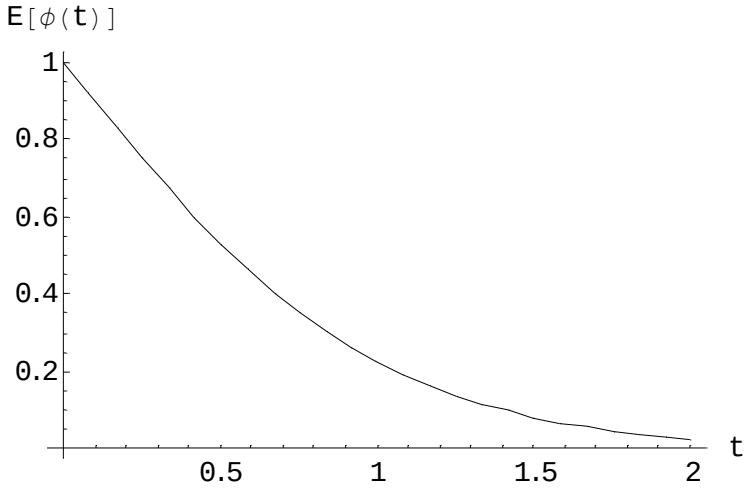


Figure 2.4: Multistate Example System State Expectation

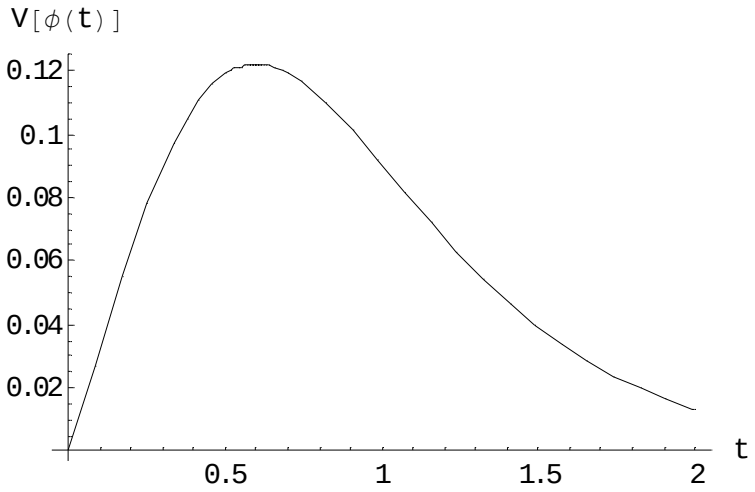


Figure 2.5: Multistate Example System State Variance

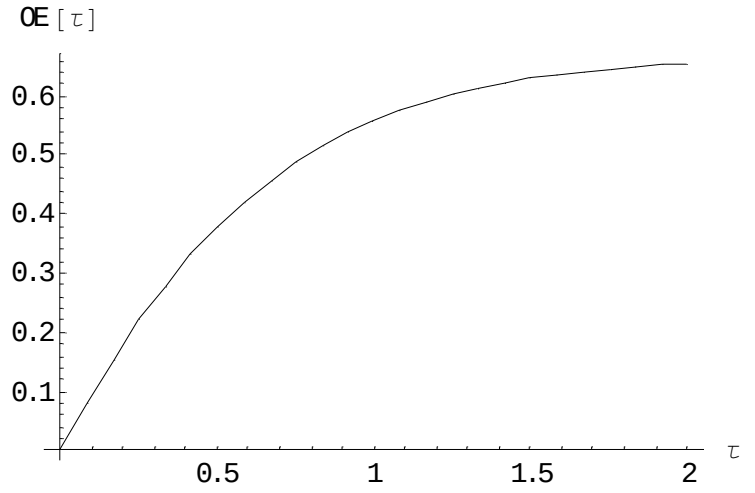


Figure 2.6: Multistate Example System Output Expectation

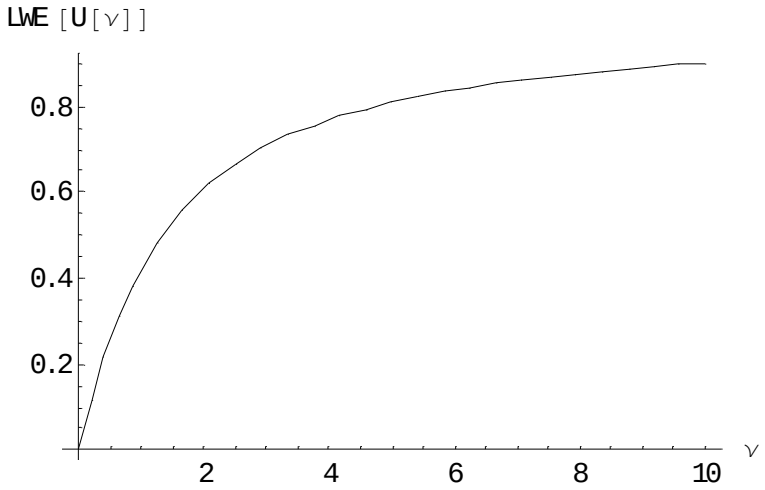


Figure 2.7: Multistate Example Lifetime-Weighted Measure

Chapter 3

SELECTION-BASED STRUCTURE FUNCTION CONSTRUCTION

3.1 Introduction

It is often possible to identify the nature of a subsystem's structure function based upon engineering knowledge — indeed, performing this task is a general requirement in Fault Tree Analysis [56]. Many types of structure functions appear again and again in real life and in the reliability literature, and for many of these structure functions probability distributions (or moments) may easily be found based on component probability distributions (or moments). The results in Tables 3.1 and 3.2 may be used to quickly compute the reliability measure building blocks identified in Table 2.1 for five common subsystem structures, hence allowing easy computation of reliability measures for large and complicated systems based on combinations of these particular subsystems; notation which is used in this dissertation for these common structures is illustrated in Figure 3.1 and Table 3.3.[1] For systems or subsystems which cannot be further subdivided or simplified, or which do not have a form given in Table 3.2, the expressions (3.1), (3.2), and (3.3) can always be used.

It is assumed in this chapter that the components are mutually statistically independent and that each component appears in only one subsystem (assuring inde-

[1] In Table 3.3, "Min Req." and "Max Req." is the minimum number of components that must be in their minimal (maximal) states for the system to be in its minimal (maximal) state. It may also be taken as the minimum number of components that must have a non-zero probability of being in their minimal (maximal) states for the system to have a non-zero probability of being in its minimal (maximal) state.

pendence between the subsystems). It is also assumed that, for multistate models, $\Omega = \{\phi(\mathbf{x}) \mid \mathbf{x} \in S\}$ (in other words, that the state space for the system is defined by the set of states that the given structure function and component state space will allow).[2]

Table 3.1: Distributions of Selected Structures

$\phi(X_1, X_2, \ldots, X_n)$	$F[s,t]$
$\max(X_1, X_2, \ldots, X_n)$	$\prod_{i=1}^{n} F_i[s,t]$
$\min(X_1, X_2, \ldots, X_n)$	$1 - \prod_{i=1}^{n}(1 - F_i[s,t])$

Table 3.2: Moments of Selected Structures

$\phi(X_1, X_2, \ldots, X_n)$	$E[\phi(t)]$	$E[\phi^2(t)]$
$\max(X_1, X_2, \ldots, X_n)$	$\int_0^M (1 - \prod_{i=1}^{n} F_i[s,t])\, ds$	$\int_0^M 2s(1 - \prod_{i=1}^{n} F_i[s,t])\, ds$
$\min(X_1, X_2, \ldots, X_n)$	$\int_0^M (\prod_{i=1}^{n}(1 - F_i[s,t]))\, ds$	$\int_0^M 2s(\prod_{i=1}^{n}(1 - F_i[s,t]))\, ds$
$\frac{1}{n}\sum_{i=1}^{n} X_i$	$\frac{1}{n}(\sum_{i=1}^{n} E[X_i(t)])$	$\frac{1}{n^2}(\sum_{i=1}^{n} E[X_i^2(t)] + 2\sum_{j=i+1}^{n}\sum_{i=1}^{n-1} E[X_i(t)]E[X_j(t)])$
$\prod_{i=1}^{n} X_i$	$\prod_{i=1}^{n} E[X_i(t)]$	$\prod_{i=1}^{n} E[X_i^2(t)]$
$1 - \prod_{i=1}^{n}(1 - X_i)$	$1 - \prod_{i=1}^{n}(1 - E[X_i(t)])$	$1 - 2\prod_{i=1}^{n}(1 - E[X_i(t)]) + \prod_{i=1}^{n}(1 - 2E[X_i(t)] + E[X_i^2(t)])$

$$E[\phi(t)] = \int_0^{M_1} \int_0^{M_2} \cdots \int_0^{M_n} \phi(x_1, x_2, \ldots, x_n)\, dF_1[x_1,t] dF_2[x_2,t] \cdots dF_n[x_n,t] \quad (3.1)$$

[2] Alternately, one may choose for multistate models to define versions of structure functions where the system state is rounded up, down, or to the nearest available state (i.e. rounded from the system state the original structure function would have specified to an adjacent *allowed* state $s \in \Omega$).

Table 3.3: Symbology for Selected Structures

$\phi(X_1, X_2, \ldots, X_n)$	Symbol	Min Req.	Max Req.
$\max(X_1, X_2, \ldots, X_n)$	max or $\uparrow$	n	1
$\min(X_1, X_2, \ldots, X_n)$	min or $\downarrow$	1	n
$\frac{1}{n}\sum_{i=1}^{n} X_i$	Σ	n	n
$\prod_{i=1}^{n} X_i$	$\prod$	1	n
$1 - \prod_{i=1}^{n}(1 - X_i)$	$\coprod$	n	1

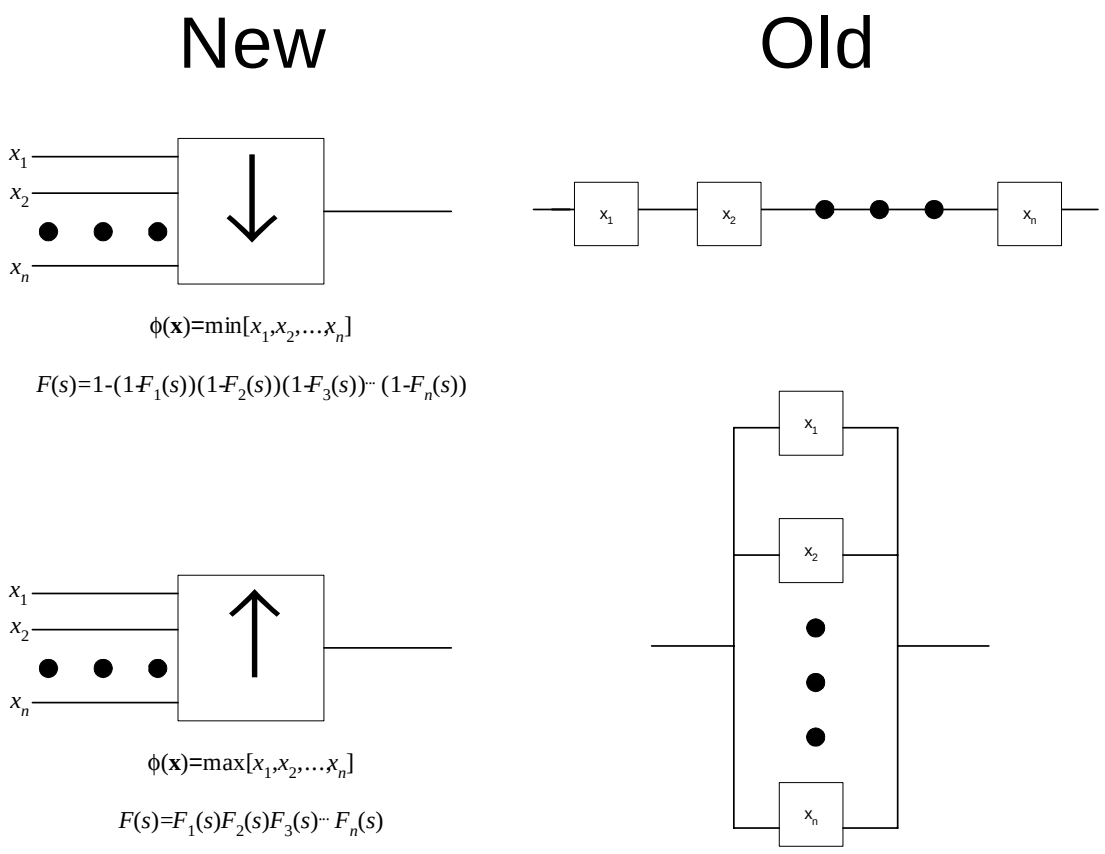


Figure 3.1: Parallel and Series Subsystem Notation

$$E[\phi^2(t)] = \int_0^{M_1} \int_0^{M_2} \cdots \int_0^{M_n} \phi^2(x_1, x_2, \ldots, x_n)\, dF_1[x_1, t] dF_2[x_2, t] \cdots dF_n[x_n, t] \tag{3.2}$$

$$F[s, t] = \int_0^{M_1} \int_0^{M_2} \cdots \int_0^{M_n} \mathrm{I}[\phi(x_1, x_2, \ldots, x_n) \leq s]\, dF_1[x_1, t] dF_2[x_2, t] \cdots dF_n[x_n, t] \tag{3.3}$$

$$\mathrm{I}[\phi(x_1, x_2, \ldots, x_n) \leq s] = \begin{cases} 1, & \phi(x_1, x_2, \ldots, x_n) \leq s \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

In Equations (3.1), (3.2), and (3.3), $dF_i[x_i, t]$ may be replaced by $f_i[x_i, t]dx_i$ when the distribution for component i is absolutely continuous and a PDF for it may be computed. Similarly, $\prod_{i=1}^{n} dF_i[x_i, t]$ may be replaced by the appropriate Riemann or Stieltjes expression in cases where the components are not mutually statistically independent, and so must be specified by a joint probability distribution.

3.2 Adding Additional Parameters

If more flexible versions of the functions listed in Table 3.3 are desired, additional parameters can be added which are ultimately specified through a finite set of questions to the customer. When creating new structure functions, it may be important know whether or not they are coherent; the expanded functions given in this section are all coherent by the definition presented in Section 1.6.

Following each definition given below is an expression for determining the unknown parameters a_i based on customer input; please note that the expressions for a_i should be modified if 1/2 is not a valid system state for the model in question.[3]

---

[3]The effects of exponents on components in coherent continuum models may be stated simply: having an $x_i^{a_i}$ term ($a_i > 0$, $a_i \neq 1$) will increase the positive effect of x_i if $a_i \in [0, 1)$ and will increase the negative effect of x_i if $a_i \in (1, \infty)$. As $a_i \to \infty^-$ or $a_i \to 0^+$, the mapping for x_i approaches a binary mapping: $x_i^{\infty^-} \to 1$ if $x_i = 1$, 0 otherwise; $x_i^{0^+} \to 0$ if $x_i = 0$, 1 otherwise.

Generalized Series: $\phi(\mathbf{x}) = \min\{x_1^{a_1}, x_2^{a_2}, \ldots, x_n^{a_n}\}$, $a_i = \ln[\phi(1/2_i, \mathbf{1})]/\ln[1/2]$

Generalized Parallel: $\phi(\mathbf{x}) = \max\{x_1^{a_1}, x_2^{a_2}, \ldots, x_n^{a_n}\}$, $a_i = \ln[\phi(1/2_i, \mathbf{0})]/\ln[1/2]$

Generalized Product: $\phi(\mathbf{x}) = x_1^{a_1} x_2^{a_2} \cdots x_n^{a_n}$, $a_i = \ln[\phi(1/2_i, \mathbf{1})]/\ln[1/2]$

Generalized Coproduct: $\phi(\mathbf{x}) = 1 - (1 - x_1)^{a_1}(1 - x_2)^{a_2} \cdots (1 - x_n)^{a_n}$
$a_i = \ln[1 - \phi(1/2_i, \mathbf{0})]/\ln[1 - 1/2]$

Generalized Average: $\phi(\mathbf{x}) = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$ (where $a_1 + a_2 + \cdots + a_n = 1$)
$a_i = \phi(\mathbf{0}, 1_i)$

Generalized k-out-of-n: $\phi(\mathbf{x}) = k$th greatest of $\{x_1^{a_1}, x_2^{a_2}, \ldots, x_n^{a_n}\}$
$a_i = \ln[\phi(1/2_i, \{k - 1$ components in state $1, n - k$ in state $0\})]/\ln[1/2]$

Constructing a subsystem from this set is therefore a three-step process (see Figure 3.2):

1. Select the function type.

2. Inquire as to whether there is symmetry in any components (defined here as components which serve the same roles and hence should have identical parameters).

3. Determine unknown parameters through a set number of questions. For example, if a continuum subsystem is generalized series and there are no symmetries, each a_i may be determined by asking, "What is the value of $\phi(\mathbf{x})$ for $(1/2_i, \mathbf{1})$?" In this case the customer's response, α_i, is related to a_i by $a_i = \frac{\ln \alpha_i}{\ln 1/2}$.

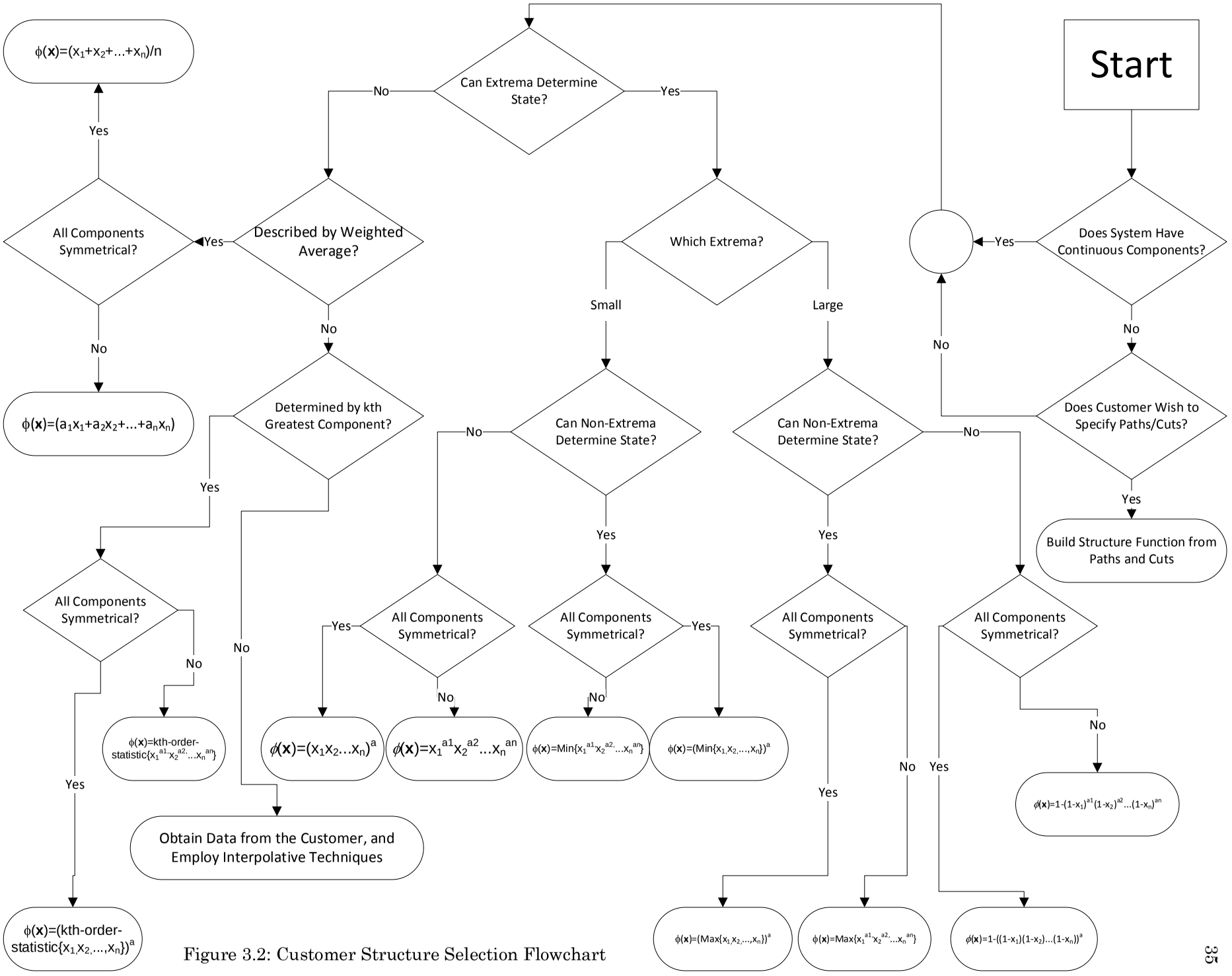


Figure 3.2: Customer Structure Selection Flowchart

3.3 Boundary Point Analysis

3.3.1 Introduction

The methods described in Section 3.2 only work when the structure function for a system or subsystem has one of the given forms.[4] The set of possible coherent structure functions is much larger; to unambiguously specify an arbitrary coherent structure function the use of "boundary points" has been recommended in the literature.

Any coherent structure function may be completely specified by the customer through an enumeration (finite, in the case of discrete models) of the upper or lower boundary points for that structure function. For discrete models, boundary points have a very simple interpretation (i.e. "For what component state vectors do a decrease in *any* component's state force a decrease of the system state?"). Algorithms outlined in [214] which can generate the structure function based on the upper or lower boundary points and which can generate the upper (or lower) boundary points based on the lower (or upper) boundary points are included in the software package which accompanies this dissertation.

As is discussed in Chapter 4, although boundary points are mathematically well-defined for continuum models, it is unclear that they have any practical value in terms of allowing the customer to specify the structure function of continuum models.

3.3.2 Notation

General

- $\mathbf{x}$ is a lower boundary point to level k if $\phi(\mathbf{x}) \geq k$ and $\mathbf{y} < \mathbf{x}$ implies that $\phi(\mathbf{y}) < k$, $k \neq 0$

- $\mathbf{x}$ is an upper boundary point to level k if $\phi(\mathbf{x}) \leq k$ and $\mathbf{y} > \mathbf{x}$ implies that

[4] Although for practical reasons many systems may be made up exclusively of modules with these common forms.

$\phi(\mathbf{y}) > k,\ k \neq M$

- L_k is the set of all lower boundary points to level k
- U_k is the set of all upper boundary points to level k
- For $\mathbf{x} \in L_k$, $L_k(\mathbf{x}) = \{(i, x_i)\ \forall x_i \neq 0\}$
- For $\mathbf{x} \in U_k$, $U_k(\mathbf{x}) = \{(i, x_i)\ \forall x_i \neq M_i\}$

Continuum Models

Unlike discrete models, attention must be paid to continuity issues when using boundary points for continuum models. Using the standard continuity definitions given below, "Lower boundary points to level k are only defined when a [continuum model] is right continuous and upper boundary points to level k are only defined when a [continuum model] is left continuous." [214]

These standard continuity definitions may be found in Montero et al. [205]:

- A continuum structure function is *left-continuous* at $\mathbf{y}$ if for each $\mathbf{x} \in S$ and for each $\epsilon > 0$, there is a $\delta > 0$ such that $|\phi(\mathbf{x}) - \phi(\mathbf{y})| < \epsilon$ whenever $\mathbf{y} - \delta\mathbf{1} < \mathbf{x} < \mathbf{y}$.
- A continuum structure function is *right-continuous* at $\mathbf{y}$ if for each $\mathbf{x} \in S$ and for each $\epsilon > 0$, there is a $\delta > 0$ such that $|\phi(\mathbf{x}) - \phi(\mathbf{y})| < \epsilon$ whenever $\mathbf{y} < \mathbf{x} < \mathbf{y} + \delta\mathbf{1}$.

Multistate Models

Since there are a finite number of boundary points in multistate cases, the following enumerative notation is sometimes used:

- L_{kj} is the jth lower boundary point to level k, $k \neq 0$, $j \in \{1, 2, \ldots, r_k\}$, $L_{kj} \in L_k$

- U_{kj} is the jth upper boundary point to level k, $k \neq M$, $j \in \{1, 2, \ldots, s_k\}$, $U_{kj} \in U_k$

Binary Models

The following notation is sometimes used in place of the general boundary point notation when the model under consideration is binary. Essentially, a minimal path P_j is the set of component indices i for which $x_i = 1$ for a given lower boundary point L_{1j}, and a minimal cut K_j is the set of component indices i for which $x_i = 0$ for a given upper boundary point U_{0j}.

Path Vector A vector $\mathbf{x}$ satisfying the condition that $\phi(\mathbf{x}) = 1$

Minimal Path Vector A path vector $\mathbf{x}$ where $\phi(\mathbf{y}) = 0$ for all $\mathbf{y} < \mathbf{x}$

Minimal Path Set The set $P_j = \{i \mid x_i = 1\}$ formed from the jth minimal path vector $\mathbf{x}$ (let r indicate the number of minimal path vectors)

Cut Vector A vector $\mathbf{x}$ satisfying the condition that $\phi(\mathbf{x}) = 0$

Minimal Cut Vector A cut vector $\mathbf{x}$ where $\phi(\mathbf{y}) = 1$ for all $\mathbf{y} > \mathbf{x}$

Minimal Cut Set The set $K_j = \{i \mid x_i = 0\}$ formed from the jth minimal cut vector $\mathbf{x}$ (let s indicate the number of minimal cut vectors)

3.3.3 Structure Function Representation by Boundary Points

General Multistate Model

This technique was introduced by Block and Savits [144]. Its basic approach is to temporarily transform the original structure function (written in terms of n multistate variables) into an equivalent structure function written as a series of binary variables. This may be done with either lower or upper boundary points.

Lower Boundary Points

The customer specifies lower boundary points to each level k, $k \neq 0$. We compute

$$\phi(\mathbf{x}) = \sum_{k=1}^{M} \phi^k(\mathbf{y}) \tag{3.5}$$

where

$$\phi^k(\mathbf{y}) = \max_{\mathbf{x} \in L_k} \min_{(i,j) \in L_k(\mathbf{x})} y_{ij} = \begin{cases} 0, & \phi(\mathbf{x}) < k \\ 1, & \phi(\mathbf{x}) \geq k \end{cases} \tag{3.6}$$

and

$$y_{ij} = \begin{cases} 0, & x_i < j \\ 1, & x_i \geq j \end{cases} \tag{3.7}$$

Upper Boundary Points

The customer specifies upper boundary points to each level k, $k \neq M$. We compute

$$\phi(\mathbf{x}) = \sum_{k=0}^{M-1} \phi^k(\mathbf{y}) \tag{3.8}$$

where

$$\phi^k(\mathbf{y}) = \min_{\mathbf{x} \in U_k} \max_{(i,j) \in U_k(\mathbf{x})} y_{ij} = \begin{cases} 0, & \phi(\mathbf{x}) \leq k \\ 1, & \phi(\mathbf{x}) > k \end{cases} \tag{3.9}$$

and

$$y_{ij} = \begin{cases} 0, & x_i \leq j \\ 1, & x_i > j \end{cases} \tag{3.10}$$

Continuum Model

This method for specifying a continuum structure function based on the lower or upper boundary points for each state was also developed by Block and Savits [198]. As there are an uncountably infinite number of sets of boundary points which may be required, this technique is of questionable value unless some general formula is available which specifies all the lower or upper boundary points as a function of k.[5]

The following expression should replace both (3.5) for lower boundary point calculations and (3.8) for upper boundary point calculations:

$$\phi(\mathbf{x}) = \int_0^1 \phi^k(\mathbf{y})\, dk \tag{3.11}$$

Binary Model

The "General Multistate Model" expressions given above are equally valid for the binary case. The following simplified expressions, however, are often used in the binary model literature.

Lower Boundary Points

Any binary coherent system can be represented as a parallel arrangement of series structures containing its minimal path sets:

$$\phi(\mathbf{x}) = \max_{j=1}^{r} \min_{i \in P_j} x_i \tag{3.12}$$

Upper Boundary Points

Any binary coherent system can be represented as a series arrangement of parallel structures containing its minimal cut sets:

$$\phi(\mathbf{x}) = \min_{j=1}^{s} \max_{i \in K_j} x_i \tag{3.13}$$

[5]This begs the question of why this conceptually complicated expression would be available if a direct expression for the structure function were not available.

Chapter 4

INTERPOLATION-BASED STRUCTURE FUNCTION CONSTRUCTION

4.1 Introduction

In discrete cases, the use of boundary points can save the customer significant amounts of time over specifying the system value for every component state vector [142]. The techniques by which one may unambiguously specify the structure function for coherent binary and multistate models based on upper or lower boundary points are well-understood, and the techniques by which one may compute system probabilities based on component probabilities through this structure function are also well-understood [214].

Similarly, work in continuum reliability models so far has either assumed the structure function to be given by formula or specified through boundary points.[1] However, as there are an infinite number of system states in the continuum case, and boundary points may have to be specified for each, it is unclear how this technique may be depended upon to reduce the effort necessary to specify the system structure function.

Using scattered data interpolation to a finite customer-supplied data set is proposed as an alternative. In addition to often producing a reasonable approximation to the true structure function based on only a limited amount of data, this approach allows the input data to be specified by the customer in a very natural way, and also

[1] In nine of the primary research papers which discuss the continuum model [195, 198, 196, 197, 212, 202, 205, 208, 210], seven also detail the use of boundary points to specify the structure function; see Section 3.3 for further discussion of boundary points.

allows for more data to be added as it becomes available without altering the basic structure of the model.[2] It is true that by interpolating a continuum model to a discrete data set one is only approximating the points at which there is no data; however, the alternative (discretizing the model) is itself an approximation, and depending on how it is performed can often be *guaranteed* to be a poorer one.

4.2 Scattered Data Interpolation

4.2.1 General Terminology

The basic problem of scattered data interpolation [230] may be stated as follows. Given N data points

$$\{\mathbf{x}_j, \phi(\mathbf{x}_j)\} \in S \times \Omega, \quad j = 1, 2, \ldots, N \tag{4.1}$$

find a function $s(\mathbf{x})$, $\forall \mathbf{x} \in S$ such that:

$$s(\mathbf{x}_j) = \phi(\mathbf{x}_j), \quad j = 1, 2, \ldots, N \tag{4.2}$$

One of the first multivariate scattered data interpolation techniques was developed in 1968 by Shepard [235]. Surveys of later techniques may be found in [229]. Although much of the scattered data interpolation literature focuses on the cases of $n = 2$ and $n = 3$, for our purposes we will often have $n > 3$; see [224] for an examination of issues specific to this situation.

As is discussed in [230], for various theoretical reasons the problem of scattered data interpolation in $n > 1$ dimensions is inherently more difficult than univariate ($n = 1$) scattered data interpolation. In fact (unlike the univariate case), none of the multivariate scattered data interpolation techniques developed before 1987 could guarantee preservation of monotonicity, even for the relatively simple $n = 2$ case [237].

[2] which would be the case if a discrete approximation were used

4.2.2 Hardy's Multiquadric Method

The method we will utilize in this chapter is a type of "radial basis function" developed in 1971 by Hardy [231]. In a 1993 survey of methods [233], it outperformed[3] all the other methods examined (volume splines, modified quadratic Shepard, volume minimum norm network, and local volume splines) for four out of six test functions.

For Hardy's method, the interpolated function $s(\mathbf{x})$ is given by

$$s(\mathbf{x}) = \sum_{i=1}^{N} c_i \sqrt{\alpha^2 + \|\mathbf{x} - \mathbf{x}_i\|^2} \tag{4.3}$$

where α^2 is a constant, and the coefficients c_i are solutions of the following system of linear equations:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1N} \\ a_{21} & a_{22} & \dots & a_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ a_{N1} & a_{N2} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_N \end{pmatrix} = \begin{pmatrix} \phi(\mathbf{x}_1) \\ \phi(\mathbf{x}_2) \\ \vdots \\ \phi(\mathbf{x}_N) \end{pmatrix} \tag{4.4}$$

where

$$a_{ij} = \left(\sqrt{\alpha^2 + \|\mathbf{x}_i - \mathbf{x}_j\|^2} \right) \quad i, j \in 1, 2, \dots, N$$

Solving the linear system of equations (4.4) to find the c_i values insures that interpolation, rather than approximation, is being performed. Michelli [232] proved that this system of linear equations is non-singular if there are no repeated points. The fact that this linear system of equations is dense [233] may limit single-precision numerical implementations to cases where $N < 300$, though techniques in [228] will allow this method to be extended to data sets of any size.

Hardy's method has no inherent limits on the number of dimensions (and hence number of components) for which it may interpolate. Other major advantages [229] include computational simplicity, speed, and continuity in all derivatives for $\alpha^2 > 0$.

---

[3] based on the size of the RMS error (see equation (4.6)) it produced for various real-world data sets

4.2.3 Effect of α^2

The parameter α^2 (referred to as R^2 in the scattered data interpolation literature) may be arbitrarily set. Typical values for α^2 are in the range $(0, 1/2]$. Although there is no systematic way to set it to an optimal value, work has been done [226, 236] that provides general guidelines.

In general, increasing the α^2 parameter has the effect of "smoothing" the interpolation that will be created, and decreasing it will cause "sharper" turns near the data points. When $\alpha^2 = 0$, the interpolation loses the continuity of its derivatives; in the one dimensional case this results in a piecewise linear interpolation. Examples of this behavior are illustrated in Figures 4.1, 4.2, and 4.3, where Hardy's method is used on the simple $n = 1$ data set given in Table 4.1.

Table 4.1: Data Set for α^2 Illustration

$\mathbf{x}$	$\phi(\mathbf{x})$
(0)	0
(1/2)	0
(1)	1

4.2.4 Shepard's Method

For the sake of comparison, Shepard's method [235] is presented below and illustrated in Figure 4.4. For our purposes, the problem with this interpolation method is that it often creates local extrema at the given data points.[4] In the case of structure functions for reliability, which we ordinarily expect to be monotonic, this is inappropriate. It is one of the first techniques developed (first published in 1968), and many variations

[4]though it is continuous only in its first derivative

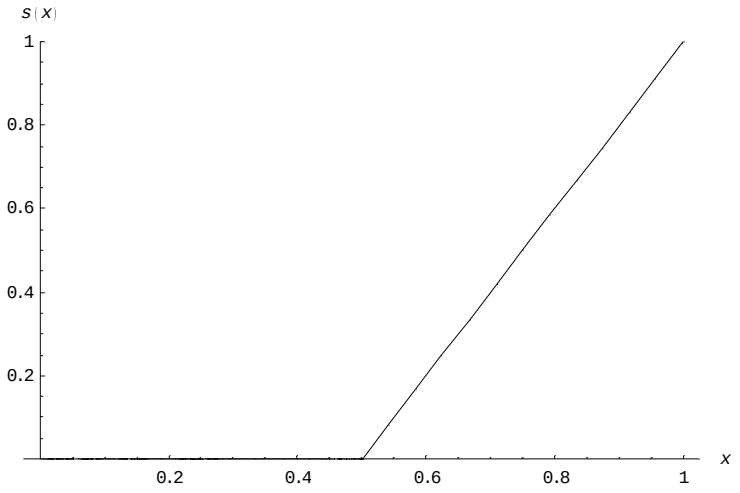


Figure 4.1: Interpolation with $\alpha^2 = 0$

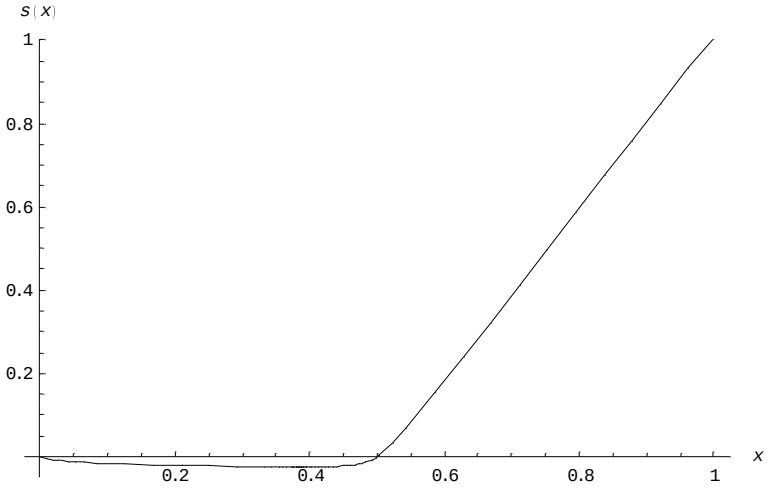


Figure 4.2: Interpolation with $\alpha^2 = 1/1000$

are available. This method does have the desirable property that the interpolated value will lie between the minimum and maximum values of the data set.

$$s(\mathbf{x}) = \frac{\sum_{i=1}^{N} \frac{\phi(\mathbf{x}_i)}{\|\mathbf{x}-\mathbf{x}_i\|^2}}{\sum_{i=1}^{N} \frac{1}{\|\mathbf{x}-\mathbf{x}_i\|^2}} \qquad \forall \mathbf{x} \neq \mathbf{x}_i,\ \forall i \in \{1, 2, \ldots, N\} \tag{4.5}$$

$$s(\mathbf{x}) = \phi(\mathbf{x}_i) \quad \forall \mathbf{x} = \mathbf{x}_i \quad \forall i \in \{1, 2, \ldots, N\}$$

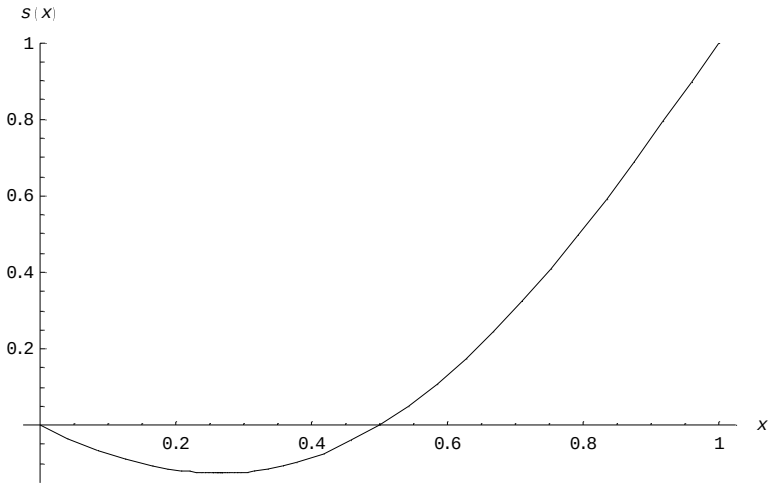


Figure 4.3: Interpolation with $\alpha^2 = 1$

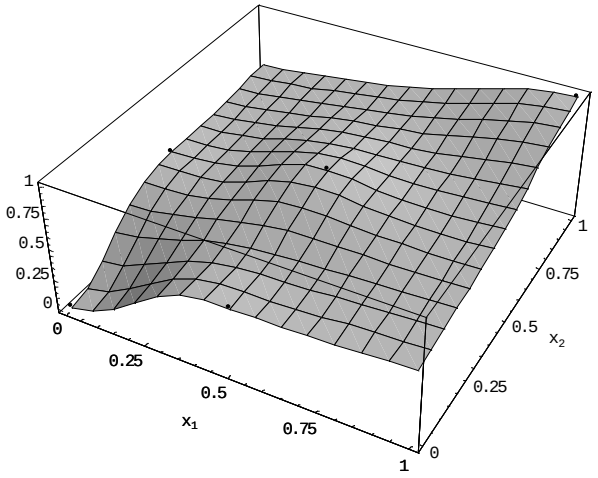


Figure 4.4: Shepard's Method Interpolation

4.2.5 Discussion

Note that if we assume the structure function we wish to approximate is coherent, we may add the extreme data points $(\mathbf{0}, 0)$ and $(\mathbf{M}, M)$ to the given data set, if they are not already present.

Research done for this chapter suggests that $\alpha^2 \approx 1/6$ may be reasonable for

continuum systems with two to four components, and higher values may be appropriate for continuum systems with greater numbers of components. In general usage, smaller values are considered when one suspects there is local variation in the underlying structure function.

Although all the interpolation methods used in this chapter may be applied to systems with any number of components, we start by illustrating their use for $n = 2$ so that the resulting $s(\mathbf{x})$ functions may be visualized. All calculations in this chapter were done on a computer with a machine round-off unit of 2.22045×10^{-16}.

4.3 Examples

4.3.1 Example 1

We assume a continuum system with $\Omega = \Omega_i = [0, 1]$, $n = 2$, and an unknown structure function $\phi(\mathbf{x})$. Five points from this structure function are given to us by the customer in Table 4.2.

Table 4.2: Example 1 Data

Component States		*System State*
x_1	x_2	ϕ
0	0	0
0	1/2	1/2
1/2	0	1/2
1/2	1/2	3/4
1	1	1

Solving the linear system (4.4) to find the c_i values yields the results given in Table 4.3. The interpolation based on these results is illustrated in Figure 4.5.

Table 4.3: c_i Values for Example 1, $\alpha^2 = 1/6$

c_1	c_2	c_3	c_4	c_5
2.034	-0.801	-0.801	-0.241	0.273

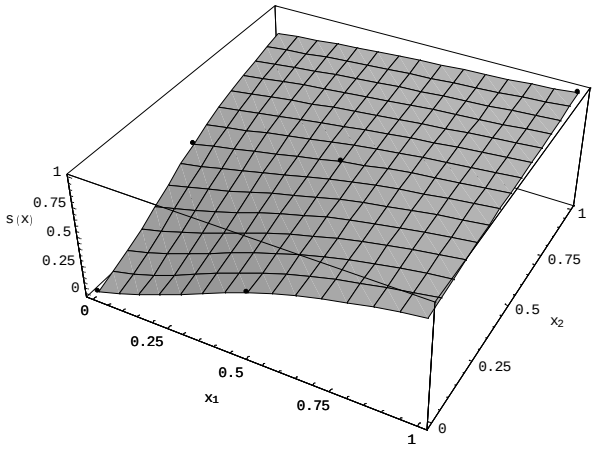


Figure 4.5: Example 1 Interpolation, $N = 5$

4.3.2 Example 2

Let us now consider a case where the underlying structure is $\phi(\mathbf{x}) = \max\{x_1, x_2\}$ as shown in Figure 4.6.

In this case, the customer specifies $N = 18$ data points to define this structure function. For the sake of this example, all but the first two state vectors $\{(0,0),0\}$ and $\{(1,1),1\}$ were randomly generated in $[0,1]^2$. Based on these data points $s(\mathbf{x})$ may be computed using Hardy's multiquadric method; this interpolation is illustrated in Figure 4.7, where dots identify the given data points.

Following the work in [233], the accuracy of this interpolation is assessed by examining the RMS deviation of our interpolated function from the underlying function, over a regular grid of $N_i \times N_j$ distinct points in the component state space (4.6).

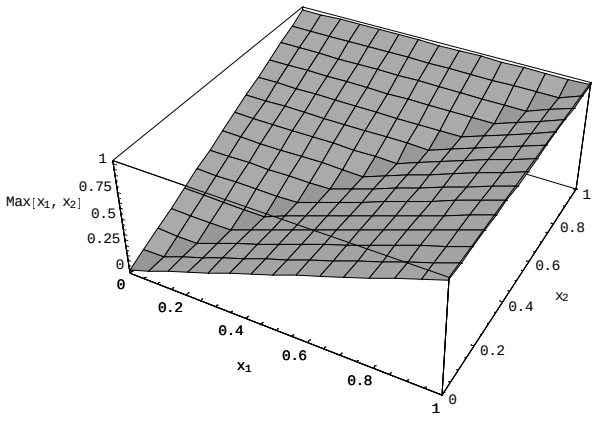


Figure 4.6: Example 2 Underlying Structure Function

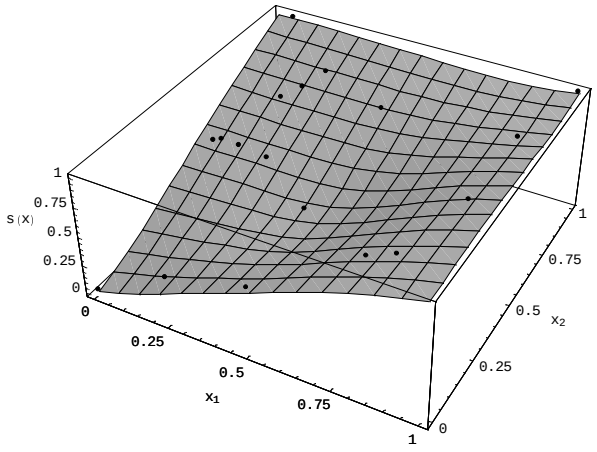


Figure 4.7: Example 2 Interpolation, $N = 18$

Throughout the remainder of this chapter, we will assume $N_i = N_j = 10$ for the purposes of RMS calculations.

$$\text{RMS} = \sqrt{\frac{\sum_{j=0}^{N_j-1}\sum_{i=0}^{N_i-1}\left[s\left(\frac{iM_1}{N_i-1},\frac{jM_2}{N_j-1}\right)-\phi\left(\frac{iM_1}{N_i-1},\frac{jM_2}{N_j-1}\right)\right]^2}{N_iN_j}} \tag{4.6}$$

For the multiquadric interpolation illustrated in Figure 4.7, RMS = 0.02400. This RMS error may be compared to the RMS error of 0.10636 obtained by using Shepard's

method [235] on the same data. Thus, for this example, the choice of interpolation method can play a large role in the estimated accuracy of the resulting interpolation.

We may also calculate $E[\phi(\mathbf{X})]$. Following an identical example in [195], we will assume that x_1 and x_2 are uniformly distributed on the interval $[0, 1]$, and that they are mutually independent. Given this information, we may (using basic theorems from probability) calculate:

$$E[s(\mathbf{X})] = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} s(x_1, x_2) f(x_1, x_2)\, dx_1 dx_2 = \int_0^1 \int_0^1 s(x_1, x_2)\, dx_1 dx_2 \qquad (4.7)$$

For this problem, $E[\phi(\mathbf{X})] = 2/3$ and $E[s(\mathbf{X})] = 0.66521$. This represents a 0.218% deviation from the true value. For this problem the value of α^2 that minimizes the RMS error is 0.22, which produces RMS = 0.02390. Thus, the heuristic selection of α^2 produced a good approximation to the optimal value. We may also examine how this interpolation improves as more data is obtained; by adding 18 more data points, randomly distributed in S, our error is reduced to RMS = 0.01539.

4.3.3 Example 3

In this example we compare three systems A, B, and C. Each of these three systems has the same set of four components, but different structure functions. For the sake of this example, we will assume that the four components in each system have known PDFs for their states, soft-truncated (see Appendix H) to lie in $[0, 1]$:

$$X_1 \sim st\text{-Weibull}(3/2,2)$$

$$X_2 \sim st\text{-Weibull}(3,4)$$

$$X_3 \sim st\text{-Beta}(25/16,1)$$

$$X_4 \sim st\text{-Normal}(3/4,1/5)$$

The components are assumed to be statistically independent. Our metric for reliability shall be the expected system state, and the structure function for each system is treated as unknown.

The customer specifies 20 data points in the component state space for each system. For the purposes of this example, all but the first two data points $\{(0,0,0,0),0\}$ and $\{(1,1,1,1),1\}$ for each system are selected randomly from $[0,1]^4$ (data sets available on the disk accompanying this dissertation, or in Appendix I). Using scattered-data interpolation ($\alpha^2 = 1/6$), we estimate the expected state of each system. Based on these results, if we were using the expected state of the system as our reliability measure we would recommend system A over system B, and system B over system C:

$$\text{Sys. A: } E[s(\mathbf{X})] = 0.624 \quad \text{Sys. B: } E[s(\mathbf{X})] = 0.545 \quad \text{Sys. C: } E[s(\mathbf{X})] = 0.444$$

Although this was not known before analysis was performed, the $\phi(\mathbf{x})$ values for the three systems were generated from known deterministic expressions:

Sys. A: $\phi(\mathbf{x}) = [\ln(x_1+1)/\ln(2) + (e^{x_2}-1)/(e-1) + x_3^3 + \sqrt{x_4}]/4$
Sys. B: $\phi(\mathbf{x}) = (x_1^{5/6} + x_2^2 + x_3^3 + x_4^{3/2})/4$
Sys. C: $\phi(\mathbf{x}) = x_1^{1/3} x_2^{4/3} x_3^{1/4} x_4^{1/5}$

We may therefore calculate the "exact" reliabilities:

$$\text{Sys. A: } E[\phi(\mathbf{X})] = 0.620 \quad \text{Sys. B: } E[\phi(\mathbf{X})] = 0.542 \quad \text{Sys. C: } E[\phi(\mathbf{X})] = 0.445$$

Thus, the recommendation to the customer based on the approximation was the same as if we would have had access to the true structure functions. This test was run with a variety of different data sets of sizes varying from 10 to 200. In no case were results returned which would have caused the wrong recommendation to be made. Furthermore, in 336 tests of these and other four-component structure functions, in no case was an error of more than 5% made in the estimate of $E[\phi(\mathbf{X})]$ when 30 data points or more were provided.

4.4 Insuring Monotonicity

4.4.1 Introduction

As was previously mentioned, insuring monotonicity is not a straightforward matter when utilizing multivariate scattered-data interpolation. However, monotonicity is a common requirement for a reliability model to be termed "coherent" [214]. It is important to note that meaningful analysis may still be performed on a reliability model, as was done in the previous two examples, without the assumption of coherence. If the coherence assumption may not be relaxed, the methods presented in this section will allow a coherent system to be generated from a coherent data set at a significant increase in computational effort.

4.4.2 Procedure and Application

If the extreme data points $\{\mathbf{0}, 0\}$ and $\{\mathbf{M}, M\}$ are elements of the customer-supplied data set, then clearly the "proper extrema" property will be satisfied. The "monotonicity" property will be retained, for a monotonic data set, by utilizing an approach we will present in this section. The "component relevance" property can then be verified by inspection, which will complete the verification of coherence for our interpolated model.

Since the only difficult part of this definition to establish is monotonicity, the remainder of this section shall be devoted to describing an interpolation scheme that preserves the monotonicity of the input data set. Our monotonicity-preserving interpolation scheme proceeds in two steps:

1. Construct an n-dimensional grid (containing up to N^n distinct intersection points) in the component state space such that each of the N known data points lies on an intersection point of the grid; use Hardy's multiquadric interpolation to compute values for each intersection point (other than those coinciding with

one of the N known data points), checking and possibly adjusting each one so that monotonicity is retained against all known or calculated grid points.

2. Use "multilinear interpolation" on this completed grid to compute $s(\mathbf{x})$ values for all points $\mathbf{x} \in S$.

We should first add the data points $\{\mathbf{0}, 0\}$ and $\{\mathbf{M}, M\}$ to the customer-supplied data set if they are not already present, and should also add to the customer-supplied data set any grid points that *must* have structure function values of 0 or M due to the monotonicity of the input data set.

Step 1

The aforementioned n-dimensional grid for the data set used in Example 2 is illustrated in Figure 4.8 (the large dots identify the customer-supplied data points). When the estimate $s(\mathbf{x})$ is calculated for each unknown point via Hardy's method, that estimate is checked against all given or previously-calculated points to insure that the monotonicity requirement is not violated; if it is, then the estimate for that point is taken to be the value closest to the original estimate which will not violate monotonicity, considering all the points it was originally checked against. Because the value assigned to each point is a function of all previously-calculated points the order of calculation can be significant, and we adopt the heuristic of calculating the unknown grid points in order of increasing distance from the centroid $(M_1/2, M_2/2, \ldots, M_n/2)$ of the component state space.

Step 2

Once feasible estimates are available for each grid point, multilinear interpolation is used to determine structure function values at all other points in the continuous component state space. Multilinear interpolation (as recommended and illustrated

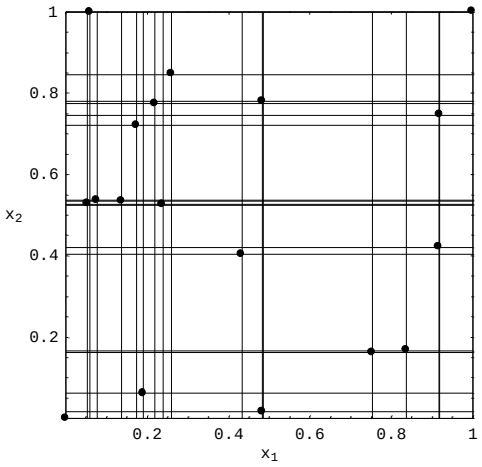


Figure 4.8: Monotonicity-Preserving Interpolation Grid for Example 2

in [277]) operates on a regular grid of known values, such as that which we now have, and preserves monotonicity [277]. The multilinear interpolation scheme is illustrated here for $n = 2$. Higher dimensions follow analogously.

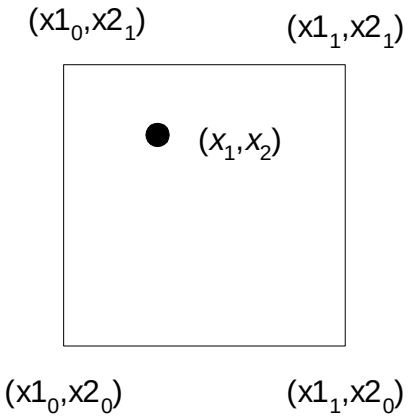


Figure 4.9: Gridded Multilinear Interpolation, $n = 2$

Assuming that $x1_0 < x1_1$, that $x2_0 < x2_1$, and that $\phi[(x1_0, x2_0)]$, $\phi[(x1_1, x2_0)]$, $\phi[(x1_0, x2_1)]$, and $\phi[(x1_1, x2_1)]$ are given (see Figure 4.9), we calculate $s_{ML}[(x_1, x_2)]$

using multilinear interpolation as follows:

$$s_{ML}[(x_1, x_2)] = \phi[(x1_0, x2_0)](1-p_1)(1-p_2) + \phi[(x1_1, x2_0)]p_1(1-p_2) + \phi[(x1_0, x2_1)](1-p_1)p_2 + \phi[(x1_1, x2_1)]p_1p_2 \tag{4.8}$$

where

$$p_1 = \frac{x_1 - x1_0}{x1_1 - x1_0}, \quad p_2 = \frac{x_2 - x2_0}{x2_1 - x2_0}$$

In Figure 4.10, the completed coherent interpolated structure function for the Example 2 data is illustrated. In this case, as there were no serious violations of monotonicity in the original interpolation, the "guaranteed-monotonicity" interpolation created with this procedure is almost indistinguishable that illustrated in Figure 4.7.

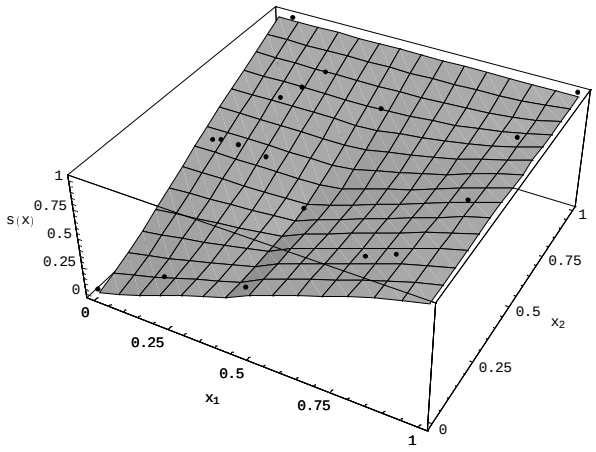


Figure 4.10: Non-Decreasing Interpolation, $N = 18$, $\alpha^2 = 1/6$

4.4.3 Coherence-Preserving Interpolation Steps

The complete procedure described in the previous section may be summarized as follows:

1. Add extrema $\{\mathbf{0}, 0\}$ and $\{\mathbf{M}, M\}$ to the customer-supplied data if they are not already present.

2. Compute grid point calculation sequence by increasing Euclidean distance from $(M_1/2, M_2/2, \ldots, M_n/2)$ (see Figure 4.11 for a sample $n = 2$ grid point calculation sequence).

3. Specify $s(\mathbf{x}) = 0$ for all grid points $\mathbf{x}$ such that $\mathbf{x} \leq \mathbf{y}_0$, where $s(\mathbf{y}_0) = 0$. Specify $s(\mathbf{x}) = M$ for all grid points $\mathbf{x}$ such that $\mathbf{y}_M \leq \mathbf{x}$, where $s(\mathbf{y}_M) = M$.

4. Calculate the multiquadric approximation to the next grid point specified in step 2, if it is not one of the customer-supplied data points. Use all customer-supplied data points in this calculation. Call this approximation $s_m(\mathbf{y}_p)$.

5. Let $(\mathbf{x}_{m1}, \mathbf{x}_{m2}, \ldots, \mathbf{x}_{ms})$ be the set of all given and calculated points such that $\mathbf{y}_p \leq \mathbf{x}_{mi}$. Calculate $v_m = \min\{s(\mathbf{x}_{m1}), s(\mathbf{x}_{m2}), \ldots, s(\mathbf{x}_{ms})\}$. Let $(\mathbf{x}_{q1}, \mathbf{x}_{q2}, \ldots, \mathbf{x}_{qr})$ be the set of all given and calculated points such that $\mathbf{x}_{qi} \leq \mathbf{y}_p$. Calculate $v_q = \max\{s(\mathbf{x}_{q1}), s(\mathbf{x}_{q2}), \ldots, s(\mathbf{x}_{qr})\}$. If $v_q \leq s_m(\mathbf{y}_p) \leq v_m$, then $s(\mathbf{y}_p) = s_m(\mathbf{y}_p)$. If $s_m(\mathbf{y}_p) > v_m$, then $s(\mathbf{y}_p) = v_m$. If $s_m(\mathbf{y}_p) < v_q$, then $s(\mathbf{y}_p) = v_q$.

6. Go to step 4, unless all grid points have been calculated.

7. Apply multilinear interpolation to all given and calculated points.

4.4.4 Discussion

Performing the two-stage interpolation procedure illustrated in Figure 4.10 was computationally intensive, and for many types of analysis would not be necessary. It should be pointed out, however, that if one's interpolated structure function preserves monotonicity from the original data set, that firm bounds may be calculated

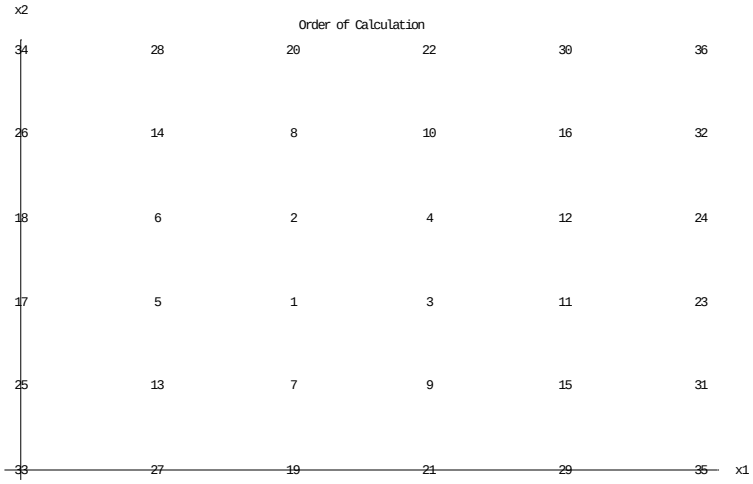


Figure 4.11: Monotonicity-Preserving Grid Point Calculation Order

on one's approximation to the expected state of the system. This may be done by defining a structure function $s_{\min}(\mathbf{x})$ which, for every $\mathbf{x}$, assumes the minimum possible value which would not violate monotonicity against the customer-supplied data set. One would similarly construct $s_{\max}(\mathbf{x})$, and could then be assured that whatever monotonicity-preserving interpolation scheme one uses to calculate $s(\mathbf{x})$, that $E[s_{\min}(\mathbf{X})] \leq E[s(\mathbf{X})] \leq E[s_{\max}(\mathbf{X})]$. See Chapter 5 for more information on this topic.

Although for the models considered in this chapter the assumption of coherence was never required to assess reliability and make engineering decisions, the creation of faster algorithms which perform scattered-data interpolations for reliability models while preserving monotonicity would be a fruitful area for further research.

Chapter 5

STRUCTURE FUNCTION PARTIAL-INFORMATION BOUNDS

5.1 Introduction

"Coherent" structure functions (whether binary, multistate, or continuum) are by definition monotonic, in the sense that an increase in a component's state may not result in a decrease of the system's state. The purpose of this chapter is to illustrate how bounds may be formed for the expected state of coherent systems defined by a finite set of customer-supplied data points. Special "$s_{\min}(\mathbf{x})$" and "$s_{\max}(\mathbf{x})$" structure functions are used to compute these bounds, given the distributions for the states of the components.

The special structure function $s_{\min}(\mathbf{x})$, which returns what at all points in S is the smallest possible system state allowed by the customer-supplied data and the assumption of monotonicity, is computed as the maximum of the given structure function values over all given data points $\mathbf{y}$ such that $\mathbf{y} \leq \mathbf{x}$.

The special structure function $s_{\max}(\mathbf{x})$, which returns what at all points in S is the greatest possible system state allowed by the customer-supplied data and the assumption of monotonicity, is computed as the minimum of the given structure function values over all given data points $\mathbf{y}$ such that $\mathbf{y} \geq \mathbf{x}$.

Using Equation (3.1) one may compute $\{E[s_{\min}(\mathbf{X})], E[s_{\max}(\mathbf{X})]\}$, which will be lower and upper bounds on the expected state of the system.

See Figure 5.1 for a sample $n = 2$ case with one non-extreme data point, and see Figures 5.2 and 5.3 for illustrations of the regions in which the structure function

values would be restricted due to monotonicity, based on a known central point.

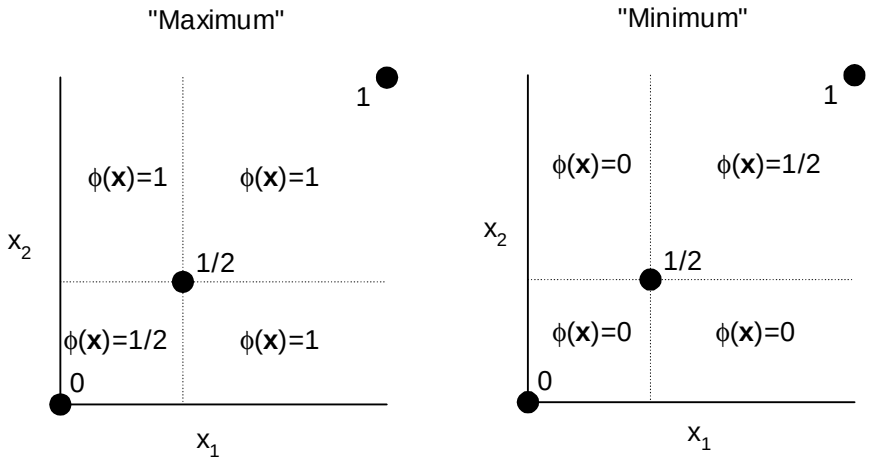


Figure 5.1: Illustration of $s_{max}(\mathbf{x})$ and $s_{min}(\mathbf{x})$ Structures

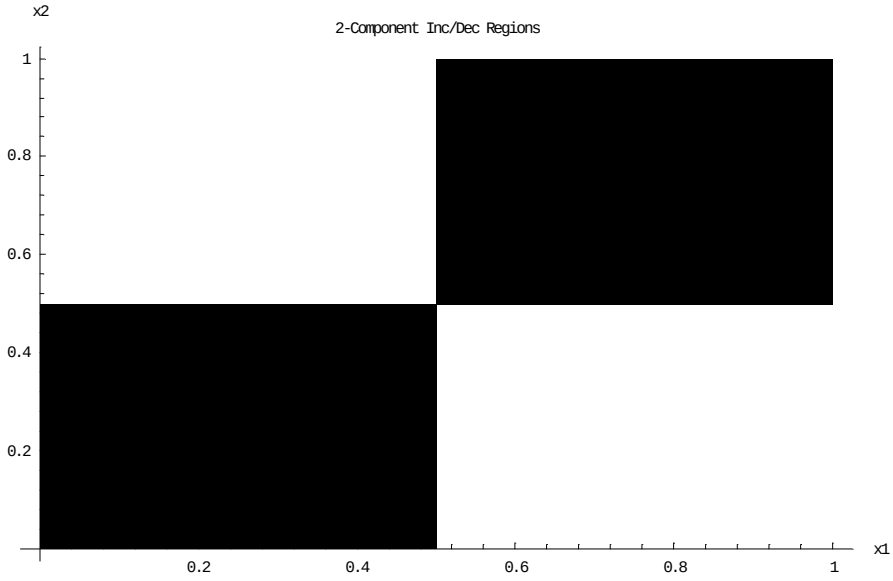


Figure 5.2: Increasing/Decreasing Regions, $n = 2$

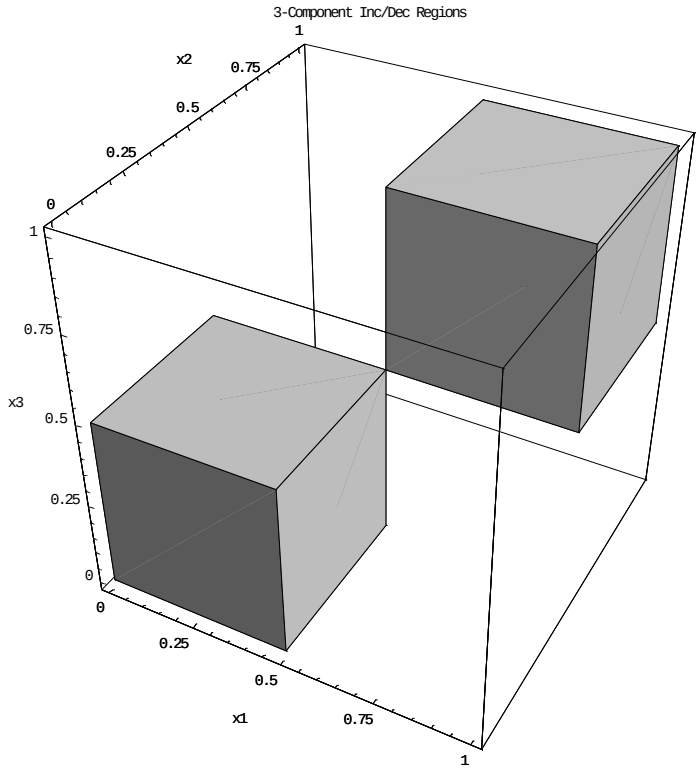


Figure 5.3: Increasing/Decreasing Regions, $n = 3$

5.2 Continuum Structure Function Bounds

5.2.1 Example 1

See Table 5.1 for the input data set used in this example. The $s_{\max}(\mathbf{x})$ and $s_{\min}(\mathbf{x})$ structure functions created from it are given in Figures 5.4 and 5.5.

Considering x_1 and x_2 to be independently and uniformly distributed on $[0, 1]$, we calculate the expected system state bounds (over the set of all monotonic structure functions which this data might be drawn from) to be $\{0.4375, 0.9375\}$. It should come as no surprise that these bounds are so wide, given how little input data was available to us.

Now, constructing the Multiquadric interpolation to this set of data (using $\alpha^2 = 1/6$), we compute the structure function illustrated in Figure 5.6, which if the original

Table 5.1: Example 1 Customer-Supplied Data

(x_1, x_2)	$\phi(x_1, x_2)$
$(0, 0)$	0
$(0, 1/2)$	$1/2$
$(1/2, 0)$	$1/2$
$(1/2, 1/2)$	$3/4$
$(1, 1)$	1

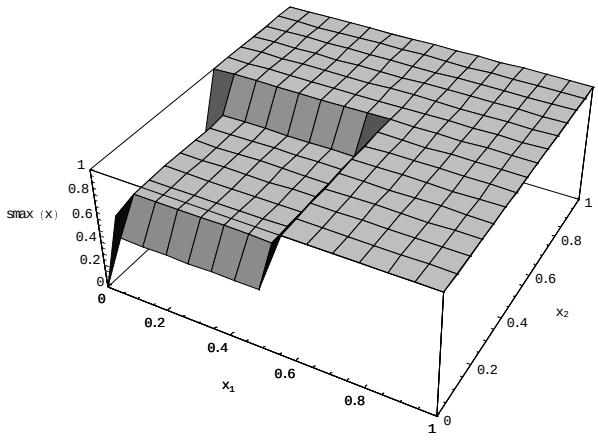


Figure 5.4: Example 1 $s_{max}(\mathbf{x})$ Structure Function

system may be assumed to be continuum and coherent, and if the resulting interpolation is non-decreasing (which it clearly is), *must* be a better approximation to the real structure function than either of the extreme structure functions $s_{\min}(\mathbf{x})$ or $s_{\max}(\mathbf{x})$. The expected state of the structure function illustrated in Figure 5.6 is 0.71351, which clearly lies between the bounds calculated for this measure.

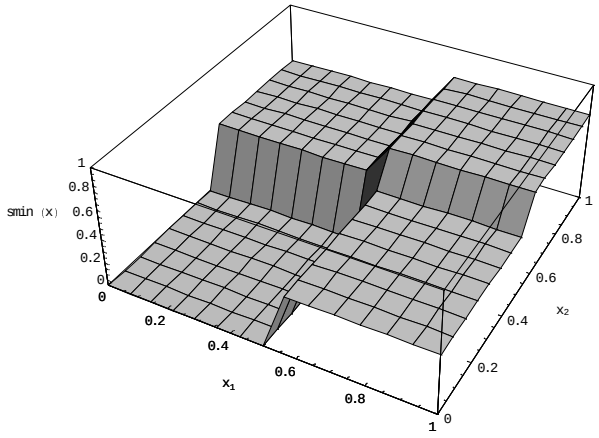


Figure 5.5: Example 1 $s_{min}(\mathbf{x})$ Structure Function

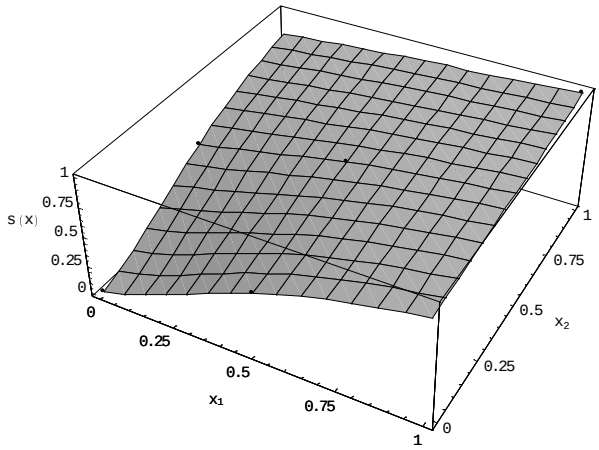


Figure 5.6: Example 1 Interpolated Structure Function

5.2.2 Example 2

Now we consider a case where we compute bounds and interpolation-based estimates for data sets of various sizes drawn from a known structure function: $\phi(\mathbf{x}) = \max\{x_1, x_2\}$ (see Figure 5.7).

Each data set includes the extreme values $\phi(0,0) = 0$ and $\phi(1,1) = 1$. All com-

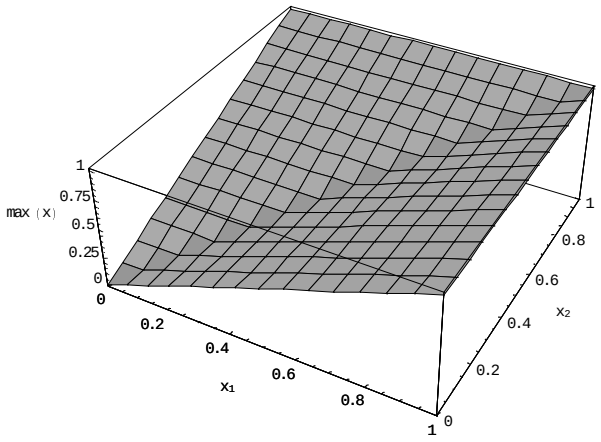


Figure 5.7: Example 2 Underlying Structure Function

ponent vectors points other than these (see Table 5.2) were chosen randomly from $[0,1]^2$.

Table 5.2: Example 2 Data Set Sizes

Data Set	Size
test2	18
test2b	36
test2c	54
test2d	72
test2e	90
test2f	108

Each component has an (independent) distribution for its state given by a Weibull distribution s-truncated to lie within $[0,1]$; the first component's Weibull distribution has parameters $(3/2, 2)$, the second has parameters $(3, 4)$. We now proceed to calculating structure function estimates and bounds, and expected system state estimates

and bounds, given these data sets and the component distributions. For the sake of comparison, we start by computing the true expected system state (using full knowledge of the underlying structure function): $E[\phi(\mathbf{X})] = 0.810897$.

We now consider expected system state bounds and estimates for this example, obtained using data sets of various sizes. Notice (see Table 5.3) how the bounds tighten as more data is added. Further insight may be gained from examining Figures 5.8-5.25, where it is apparent how all the computed structure functions approach the true structure function as more data is added.

Table 5.3: Example 2 Expected System States

Data Set	Bounds	Estimate
test2	$\{0.704411, 0.927995\}$	0.812209
test2b	$\{0.725887, 0.866069\}$	0.791326
test2c	$\{0.753227, 0.851401\}$	0.812723
test2d	$\{0.776228, 0.846707\}$	0.811123
test2e	$\{0.782112, 0.844489\}$	0.816339
test2f	$\{0.783856, 0.840450\}$	0.815713

5.3 Example 2 Figures

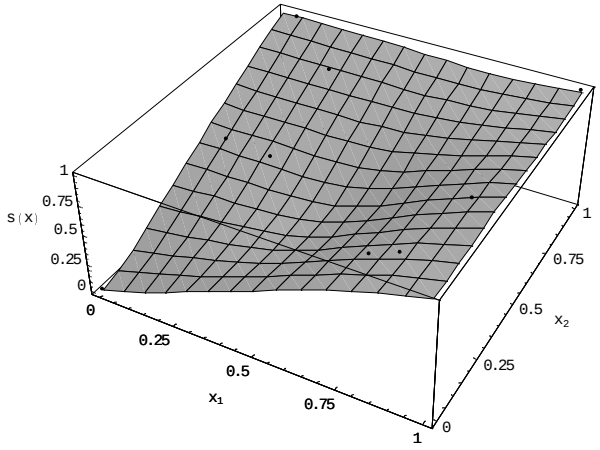


Figure 5.8: Interpolation for "test2"

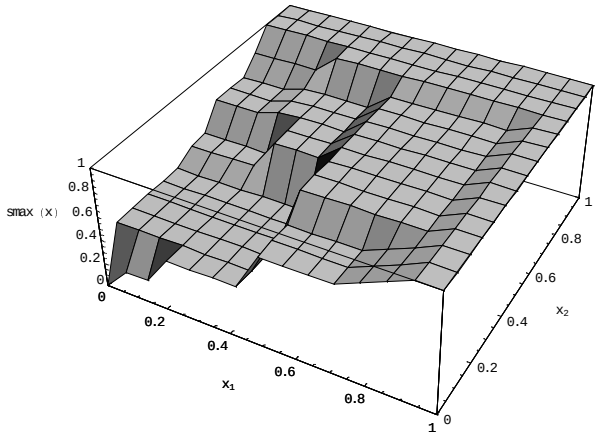


Figure 5.9: $s_{max}(\mathbf{x})$ Structure Function for "test2"

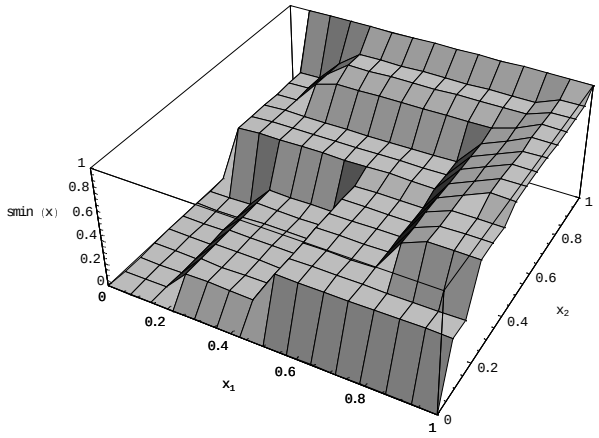


Figure 5.10: $s_{min}(\mathbf{x})$ Structure Function for "test2"

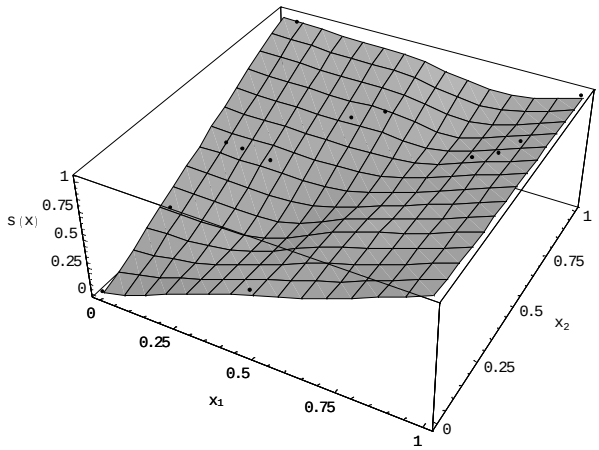


Figure 5.11: Interpolation for "test2b"

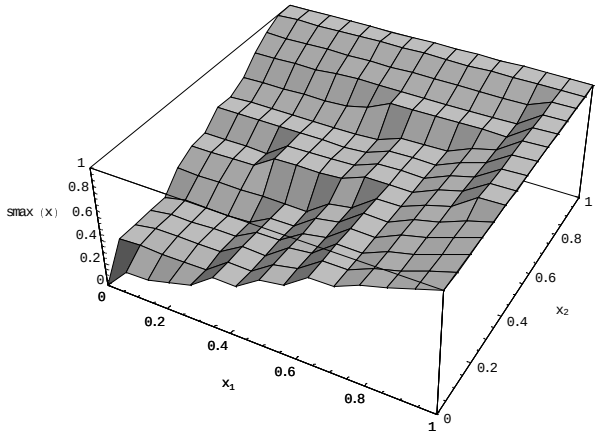


Figure 5.12: $s_{max}(\mathbf{x})$ Structure Function for "test2b"

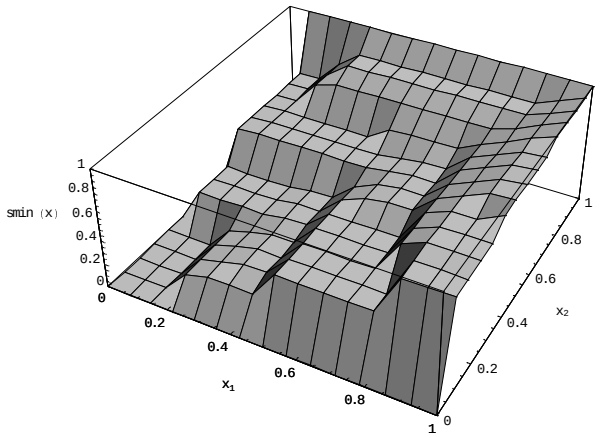


Figure 5.13: $s_{min}(\mathbf{x})$ Structure Function for "test2b"

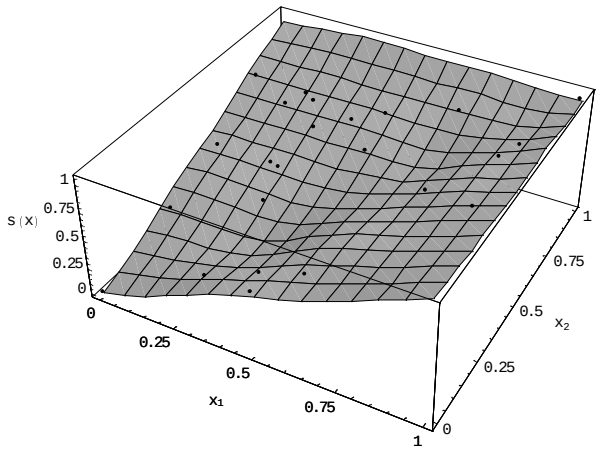


Figure 5.14: Interpolation for "test2c"

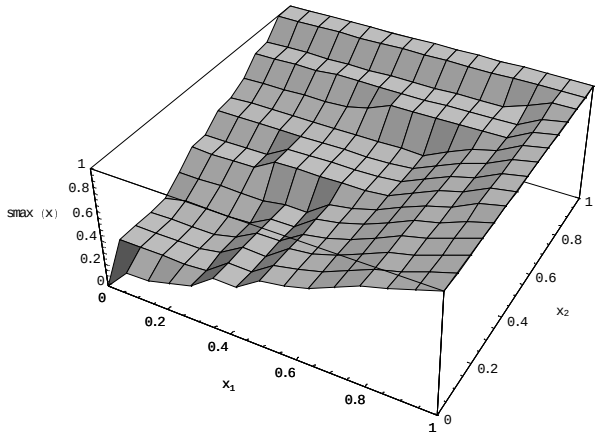


Figure 5.15: $s_{max}(\mathbf{x})$ Structure Function for "test2c"

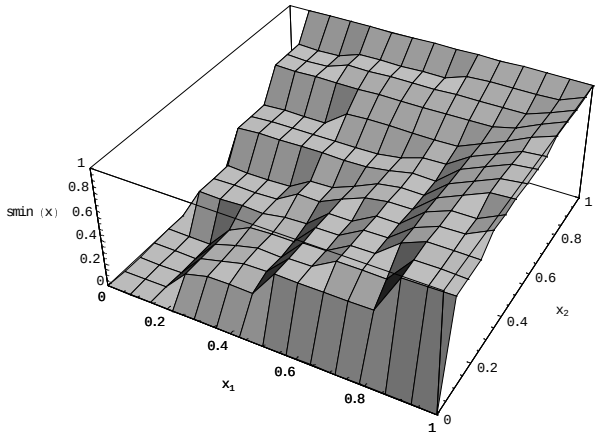


Figure 5.16: $s_{min}(\mathbf{x})$ Structure Function for "test2c"

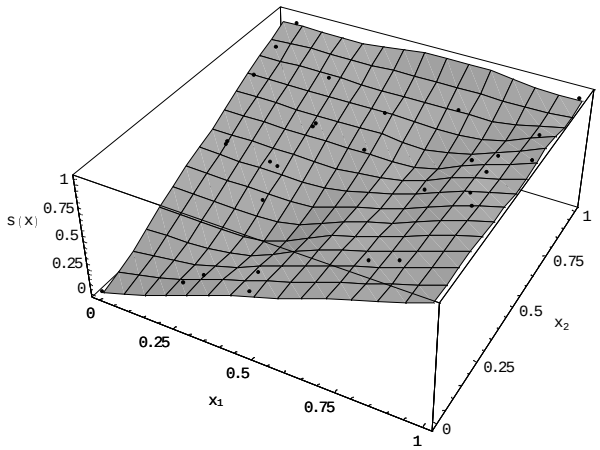


Figure 5.17: Interpolation for "test2d"

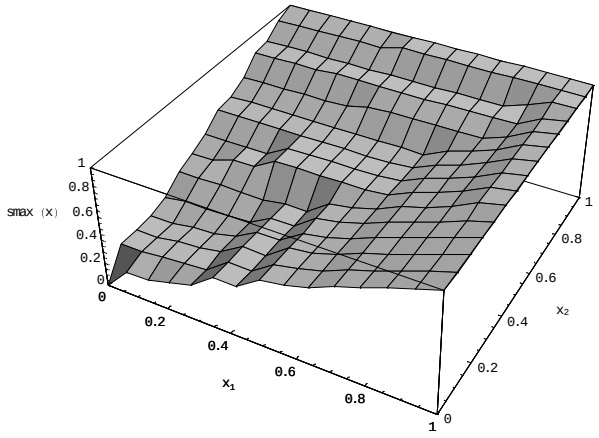


Figure 5.18: $s_{max}(\mathbf{x})$ Structure Function for "test2d"

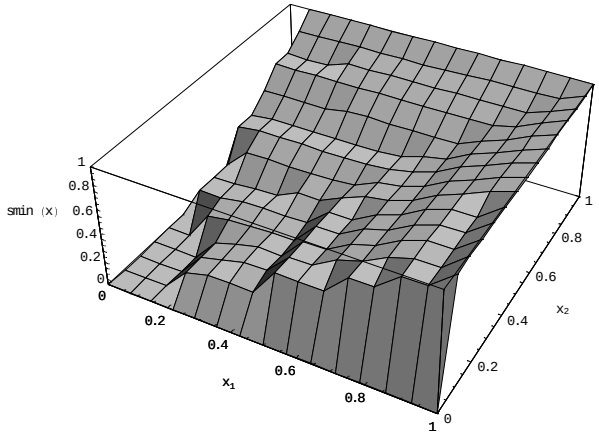


Figure 5.19: $s_{min}(\mathbf{x})$ Structure Function for "test2d"

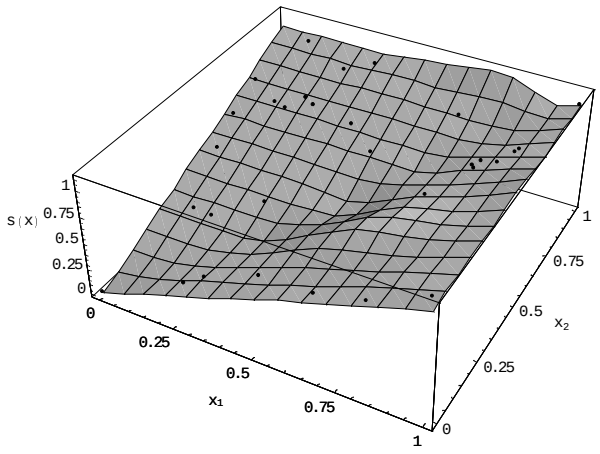


Figure 5.20: Interpolation for "test2e"

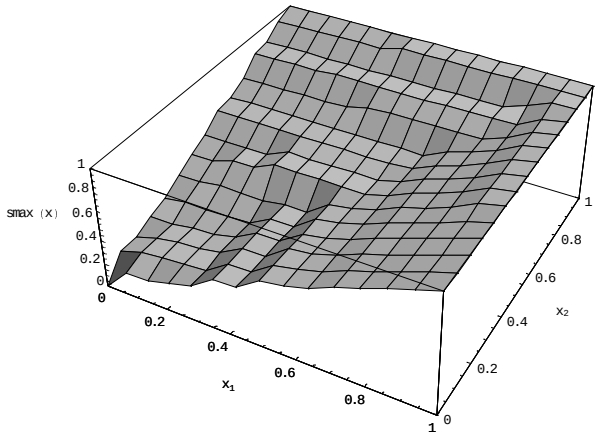


Figure 5.21: $s_{max}(\mathbf{x})$ Structure Function for "test2e"

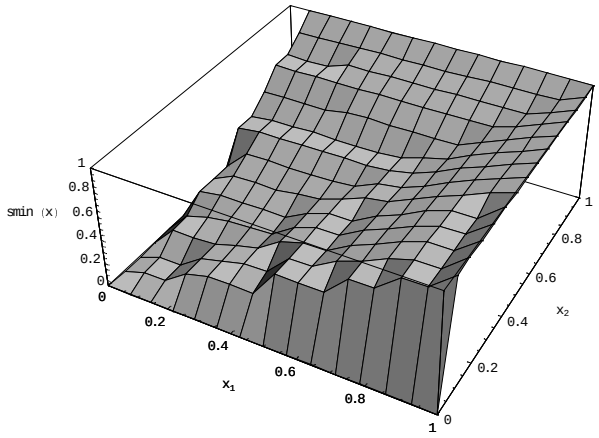


Figure 5.22: $s_{min}(\mathbf{x})$ Structure Function for "test2e"

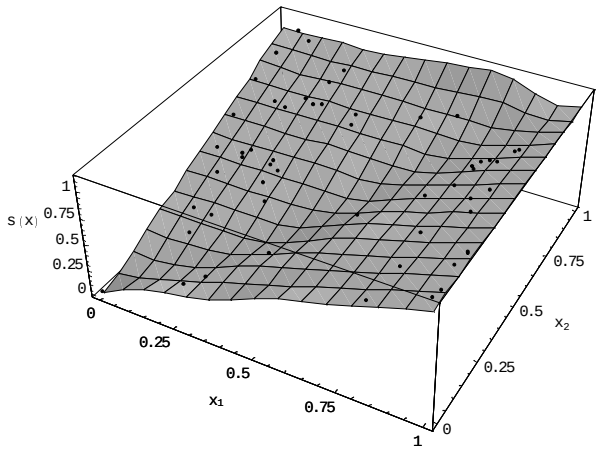


Figure 5.23: Interpolation for "test2f"

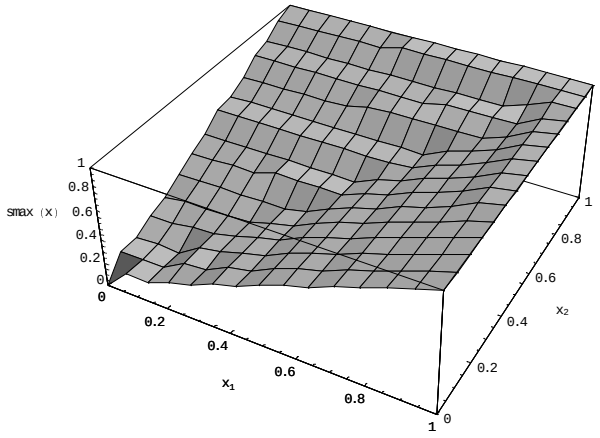


Figure 5.24: $s_{max}(\mathbf{x})$ Structure Function for "test2f"

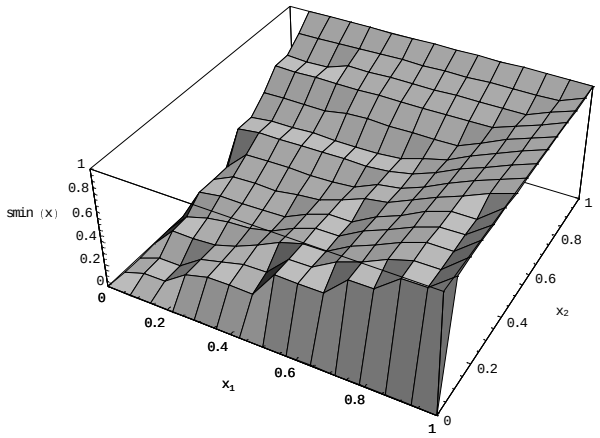


Figure 5.25: $s_{min}(\mathbf{x})$ Structure Function for "test2f"

5.4 Discrete Structure Function Bounds

We define a new coherent structure function, "Example 3," in Table 5.4 (the component state probabilities are given in Table 5.6).

Table 5.4: Example 3 Full Structure Function

$\mathbf{x}$	$\phi(\mathbf{x})$	$\mathbf{x}$	$\phi(\mathbf{x})$
(0,0)	0	(3,2)	3
(1,0)	0	(4,2)	3
(2,0)	1	(5,2)	3
(3,0)	1	(0,3)	0
(4,0)	1	(1,3)	3
(5,0)	1	(2,3)	4
(0,1)	0	(3,3)	4
(1,1)	0	(4,3)	4
(2,1)	2	(5,3)	5
(3,1)	2	(0,4)	0
(4,1)	2	(1,4)	3
(5,1)	2	(2,4)	4
(0,2)	0	(3,4)	4
(1,2)	3	(4,4)	5
(2,2)	3	(5,4)	5

Now, let us assume that we have access only to a subset of this table, so that the structure function values are known only for certain component vectors (see Table 5.5).

Using Tables 5.4 and 5.6, the "true" expected system state is computed to be 2.295. If rather than Table 5.4 we would have had access only to the partial set given in Table 5.5, we could have computed upper and lower bounds on the expected

Table 5.5: Example 3 Customer-Supplied Data

$\mathbf{x}$	$\phi(\mathbf{x})$
(0,0)	0
(2,0)	1
(5,1)	2
(5,2)	3
(0,3)	0
(1,3)	3
(3,3)	4
(5,3)	5
(0,4)	0
(5,4)	5

Table 5.6: Example 3 Component Probabilities

j	$P[X_1 = j]$	$P[X_2 = j]$
0	0.10	0.20
1	0.20	0.20
2	0.30	0.30
3	0.20	0.10
4	0.15	0.20
5	0.05	0.00

system state (in a manner consistent with the earlier sections of this chapter) as: $\{1.475, 2.690\}$.

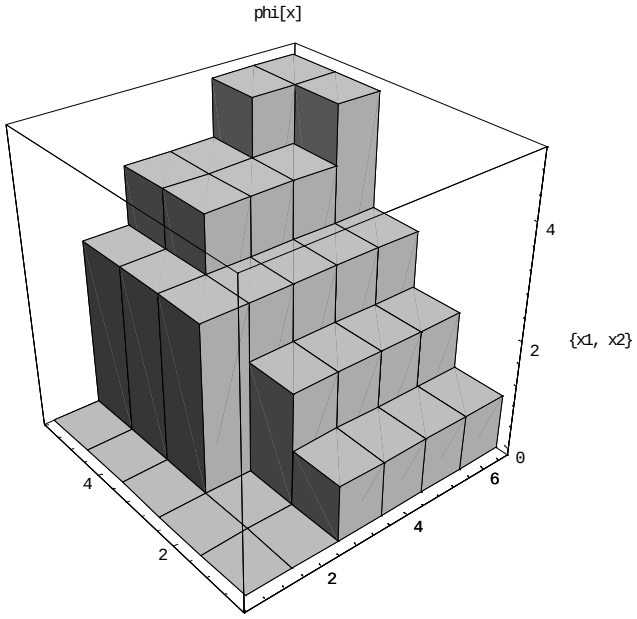


Figure 5.26: Example 3 Full Structure Function

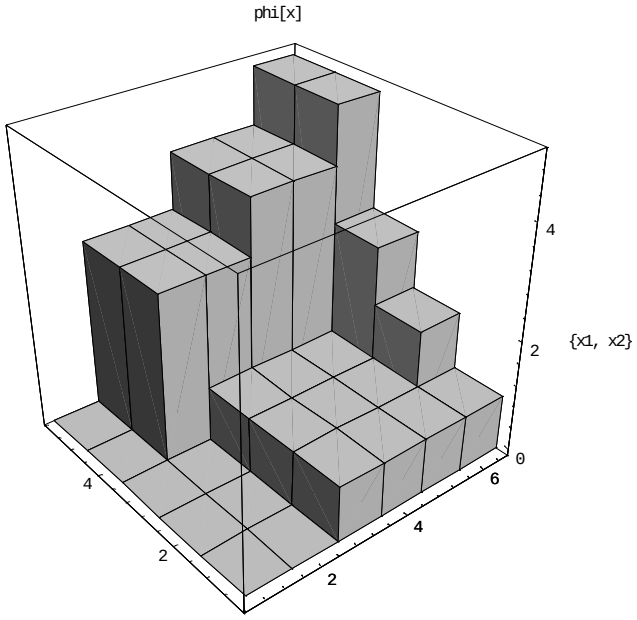


Figure 5.27: Example 3 $s_{min}(\mathbf{x})$ Bound

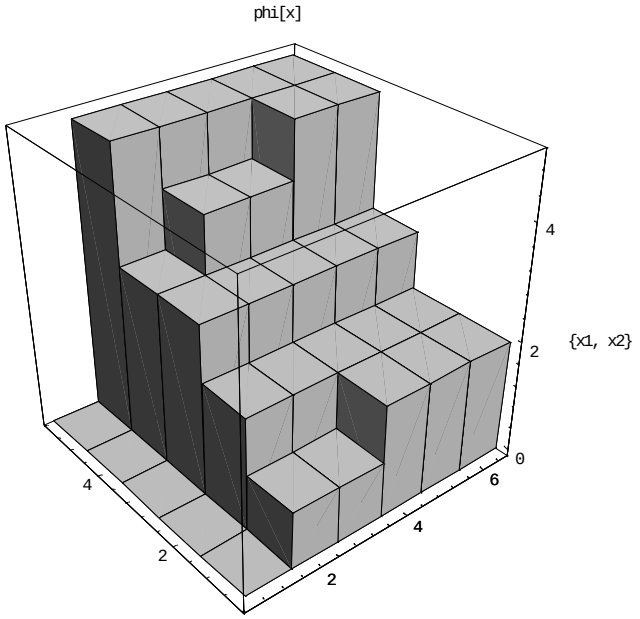


Figure 5.28: Example 3 $s_{max}(\mathbf{x})$ Bound

Chapter 6

MARKOV MODELING FOR MULTISTATE MODELS

6.1 Introduction

Frequently, an important step in multistate modeling is to compute the probabilities of the system (or its components) being in any one of its possible states.[1] As the state of a multistate system or component is a stochastic process with time as a continuous index set $t \in [0, \infty)$, and as the state space for this stochastic process is a discrete, finite set, it may be reasonable to model with a continuous time, discrete state Markov chain if one can assume the Markov property holds (a stochastic process exhibiting the Markov property has the characteristic that "the probability of any particular future behavior of the process, when its current state is known exactly, is not altered by additional knowledge concerning its past behavior" [267]).

This chapter presents expressions for the time-dynamic state probabilities, as functions of the number of states and their (stationary) transition rates, for several common classes of continuous time, discrete state Markov chains that have value in multistate reliability modeling.[2] For information on solving more general classes of Markov chains see Karlin and Taylor [267]; these general solutions are incorporated into the software which accompanies this dissertation. For other work which treats the multistate reliability model as a Markov chain, see Yang and Xue [194] or

---

[1] Of course, if the system is comprised of components with known or calculable distributions, one would calculate the system's distribution from its components' distributions and structure function rather than modeling the system distribution separately.

[2] Please see Appendix F for a summary of distribution classification and closure schemes which have been proposed in the reliability literature.

Mohamed [132].

For notational convenience, and without loss of generality, the ordinal numbers for the multistate reliability model system states ($\Omega = \{0, 1, 2, \ldots, M\}$ rather than $\Omega = \{\phi_0, \phi_1, \ldots, \phi_m\}$) are frequently used in the discussion to follow. Additionally, although Markov modeling is equally applicable to both components and systems, the notation used will be for systems rather than components.

The systems which will be considered in this chapter are non-repairable, begin in their maximal states M at $t = 0$, and have exponential sojourn times in each state. We define $P_{ij}(h)$ as the probability that a system currently in state i will be in state j at a time h time units in the future, and define $P_n(t)$ as the probability that the system will be in state n at time t.

6.2 Transitions to Adjacent States Only

In this section we consider the case where the system may not skip states as it degrades (see Figure 6.1). It begins in state M and remains in each state i ($i \neq 0$) for a length of time determined by an exponential distribution with finite parameter μ_i. Immediately after its sojourn in each state i, the system moves to state $i-1$ (the average time the system will spend in each state i is thus $1/\mu_i$). The lowest state, $i = 0$, is the only absorbing state. Therefore, we have the following constraints

$$P_M(0) = 1 \tag{6.1a}$$

$$\begin{cases} 0 < \mu_i < \infty, & 0 < i \leq M \\ \mu_i = 0, & i = 0 \end{cases} \tag{6.1b}$$

and may write the Markov transition matrix as follows:

$$\begin{bmatrix} 0 & \mu_1 & 0 & \cdots & 0 \\ 0 & -\mu_1 & \mu_2 & \cdots & 0 \\ 0 & 0 & -\mu_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \mu_M \\ 0 & 0 & 0 & \cdots & -\mu_M \end{bmatrix} \tag{6.2}$$

We assume that $P_{M+1}(t) = 0$, as the system may not attain a state higher than M as a decreasing process beginning in state M. Note that $P_M(t) = e^{-\mu_M t}$, as the system begins in state M and will remain in that initial state for a length of time determined by an exponential distribution with parameter μ_M.

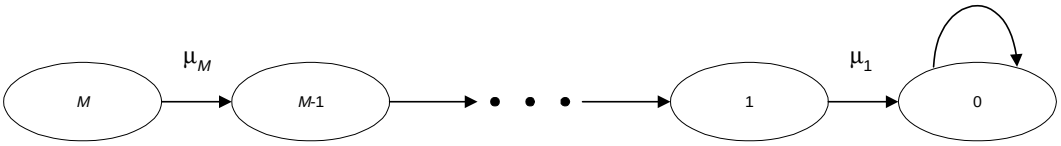


Figure 6.1: One-Step Transition Diagram

6.2.1 Identical Transition Rates

Let us first consider the case where $\mu_i = \mu_j$ for all $i \neq j$ $(i, j \in \{1, 2, \ldots, M\})$. As the random variables for the time spent in each state are exponential, identical, and independent, the Erlang distribution may be used to find the desired solutions. The PDF of the time to reach state 0 may be written as

$$f_{W_0}(t) = \frac{\mu^M}{(M-1)!} t^{M-1} e^{-\mu t} \quad t \geq 0 \tag{6.3}$$

Similarly, the probabilities of being in each state as a function of time may be written as

$$p[j,t] = \begin{cases} \frac{(\mu t)^{M-j} e^{-\mu t}}{(M-j)!} & \text{for } j = M, M-1, \ldots, 1 \\ 1 - \sum_{j=1}^{M} p[j,t] & \text{for } j = 0 \end{cases} \tag{6.4}$$

6.2.2 Distinct Transition Rates

Let us now consider the case where $\mu_i \neq \mu_j$ for all $i \neq j$ $(i, j \in \{1, 2, \ldots, M\})$. For such a process we have the following infinitesimal transitions.

$$\lim_{h \to 0} P_{ij}(h) = \begin{cases} 1 - \mu_i h, & j = i \\ \mu_i h, & j = i - 1 \end{cases} \tag{6.5}$$

From ordinary continuous-time Markov chain theory, we may write the Kolmogorov forward equation for this system:

$$P_n(t + h) = P_{n+1}(t)\mu_{n+1}h + P_n(t)(1 - \mu_n h) \tag{6.6}$$

One may rearrange, divide by h, and let $h \to 0$ to obtain:

$$P_n'(t) = P_{n+1}(t)\mu_{n+1} - P_n(t)\mu_n \tag{6.7}$$

We define the following, where $\mathcal{L}$ is the Laplace transform operator (see Appendix G)

$$\mathcal{L}_i(z) = \mathcal{L}[P_i(t)] \tag{6.8}$$

and note that

$$\mathcal{L}_M(z) = \int_0^\infty e^{-zt} P_M(t)\, dt = \frac{1}{\mu_M + z} \tag{6.9}$$

Taking the Laplace transform of the equation for $P_n'(t)$ yields (for $n < M$):

$$z\mathcal{L}_n(z) = \mu_{n+1}\mathcal{L}_{n+1}(z) - \mu_n \mathcal{L}_n(z) \tag{6.10}$$

which simplifies to

$$\mathcal{L}_n(z) = \frac{\mu_{n+1}}{\mu_n + z}\mathcal{L}_{n+1}(z) \tag{6.11}$$

Applying this equation recursively, and using the expression for $\mathcal{L}_M(z)$, we obtain:

$$\mathcal{L}_n(z) = \frac{\mu_M \mu_{M-1} \cdots \mu_{n+1}}{(\mu_M + z)(\mu_{M-1} + z) \cdots (\mu_n + z)} \tag{6.12}$$

To be able to easily take the inverse Laplace transform so that a simple, closed-form solution can be found, we apply the partial fractions expansion (this is the point at which the "distinct transition rates" assumption simplifies matters):

$$\mathcal{L}_n(z) = \mu_M \mu_{M-1} \cdots \mu_{n+1} \left(\frac{A_{M,n}}{\mu_M + z} + \frac{A_{M-1,n}}{\mu_{M-1} + z} + \cdots + \frac{A_{n,n}}{\mu_n + z} \right) \tag{6.13}$$

One may compute the A constants by examining:

$$\frac{A_{M,n}}{\mu_M + z} + \frac{A_{M-1,n}}{\mu_{M-1} + z} + \cdots + \frac{A_{n,n}}{\mu_n + z} = \frac{1}{(\mu_M + z)(\mu_{M-1} + z) \cdots (\mu_n + z)} \tag{6.14}$$

Multiply both sides by $\mu_k + z$ (where $k \in \{M, M-1, \ldots, n\}$), and set $z = -\mu_k$. This yields:

$$A_{k,n} = \frac{1}{(\mu_M - \mu_k)(\mu_{M-1} - \mu_k) \cdots (\mu_{k+1} - \mu_k)(\mu_{k-1} - \mu_k) \cdots (\mu_n - \mu_k)} \tag{6.15}$$

Now, taking the inverse Laplace transform of the above expression for $\mathcal{L}_n(z)$, utilizing the fact that the inverse Laplace transform is a linear operator, and taking advantage of the fact that

$$\mathcal{L}^{-1} \left[\frac{1}{a + z} \right] = e^{-at} \tag{6.16}$$

we obtain the final result:

for $n = M$

$$P_n(t) = e^{-\mu_M t} \tag{6.17}$$

for $n = M-1, M-2, \ldots, 0$

$$P_n(t) = \mu_{n+1} \mu_{n+2} \ldots \mu_M (A_{n,n} e^{-\mu_n t} + A_{n+1,n} e^{-\mu_{n+1} t} + \cdots + A_{M,n} e^{-\mu_M t}) \tag{6.18}$$

$$A_{k,n} = \frac{1}{(\mu_M - \mu_k)(\mu_{M-1} - \mu_k) \cdots (\mu_{k+1} - \mu_k)(\mu_{k-1} - \mu_k) \cdots (\mu_n - \mu_k)} \tag{6.19}$$

If the utility per unit time associated with state i is ϕ_i, we may consider the average total utility delivered by the product over its lifetime as $\sum_{i=1}^{m} \frac{\phi_i}{\mu_i}$. Also, by

basic theorems for expectations of sums of random variables, and from theorems for variances of sums of independent random variables, we can calculate the mean and variance of the time when the system first reaches state k, which we call W_k:

$$E[W_k] = \sum_{i=k+1}^{M} \frac{1}{\mu_i} \tag{6.20a}$$

$$V[W_k] = \sum_{i=k+1}^{M} \frac{1}{\mu_i^2} \tag{6.20b}$$

Examples

For $M = 1$

$$p[1,t] = e^{-t\mu_1} \tag{6.21}$$

$$p[0,t] = 1 - e^{-t\mu_1} \tag{6.22}$$

For $M = 2$

$$p[2,t] = e^{-t\mu_2} \tag{6.23}$$

$$p[1,t] = \mu_2 \left(\frac{e^{-t\mu_2}}{\mu_1 - \mu_2} + \frac{e^{-t\mu_1}}{\mu_2 - \mu_1} \right) \tag{6.24}$$

$$p[0,t] = 1 + \frac{\mu_2 e^{-t\mu_1}}{\mu_1 - \mu_2} + \frac{\mu_1 e^{-t\mu_2}}{\mu_2 - \mu_1} \tag{6.25}$$

For $M = 3$

$$p[3,t] = e^{-t\mu_3} \tag{6.26}$$

$$p[2,t] = \mu_3 \left(\frac{e^{-t\mu_3}}{\mu_2 - \mu_3} + \frac{e^{-t\mu_2}}{\mu_3 - \mu_2} \right) \tag{6.27}$$

$$p[1,t] = \mu_2\mu_3 \left(\frac{e^{-t\mu_3}}{(\mu_1 - \mu_3)(\mu_2 - \mu_3)} + \frac{e^{-t\mu_2}}{(\mu_1 - \mu_2)(\mu_3 - \mu_2)} + \frac{e^{-t\mu_1}}{(\mu_2 - \mu_1)(\mu_3 - \mu_1)} \right) \tag{6.28}$$

$$p[0,t] = 1 - (p[3,t] + p[2,t] + p[1,t]) \tag{6.29}$$

6.3 Transitions to Any Lower State

Now let us consider the case where the system may proceed from each state to any lower state in one transition. If the current state of the system is j, and the next state of the system will be k, we say that the length of time the system spends in state j before it transitions to state k is given by an exponential distribution with parameter $\mu_{j,k}$. We define the transition rate out of each state as $\mu_j = \sum_{i=0}^{j-1} \mu_{j,i}$ (thus, the average time the system spends in a state j before transition will be $1/\mu_j$). One may readily see, based on the nature of this problem, that $\mu_{j,k} = 0$ for $k > j$ and that $\mu_0 = 0$. The Markov transition matrix may be stated as follows:

$$\begin{bmatrix} 0 & \mu_{1,0} & \mu_{2,0} & \cdots & \mu_{M-1,0} & \mu_{M,0} \\ 0 & -\mu_1 & \mu_{2,1} & \cdots & \mu_{M-1,1} & \mu_{M,1} \\ 0 & 0 & -\mu_2 & \cdots & \mu_{M-1,2} & \mu_{M,2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & -\mu_{M-1} & \mu_{M,M-1} \\ 0 & 0 & 0 & \cdots & 0 & -\mu_M \end{bmatrix} \tag{6.30}$$

6.3.1 Examples

Applying the general theory [267], we compute the following examples:

For $M = 1$

$$p[1,t] = e^{-t\mu_1} \tag{6.31}$$

$$p[0,t] = 1 - e^{-t\mu_1} \tag{6.32}$$

For $M = 2$

$$p[2,t] = e^{-t\mu_2} \tag{6.33}$$

$$p[1,t] = \frac{\mu_{2,1}(e^{-t\mu_1} - e^{-t\mu_2})}{\mu_2 - \mu_1} \tag{6.34}$$

$$p[0,t] = 1 + \frac{\mu_{2,1}}{\mu_1 - \mu_2}e^{-t\mu_1} + \frac{\mu_{2,0} - \mu_1}{\mu_1 - \mu_2}e^{-t\mu_2} \tag{6.35}$$

For $M = 3$

$$p[3,t] = e^{-t\mu_3} \tag{6.36}$$

$$p[2,t] = \frac{\mu_{3,2}}{\mu_3 - \mu_2}(e^{-t\mu_2} - e^{-t\mu_3}) \tag{6.37}$$

$$\begin{aligned} p[1,t] = &\left(\frac{\mu_2\mu_{3,1} - \mu_3\mu_{3,1} + \mu_{2,1}\mu_{3,2}}{(\mu_1 - \mu_3)(\mu_2 - \mu_3)}\right)e^{-t\mu_3} \\ &+ \left(\frac{\mu_{2,1}\mu_{3,2}}{(\mu_1 - \mu_2)(\mu_3 - \mu_2)}\right)e^{-t\mu_2} \\ &+ \left(\frac{\mu_2\mu_{3,1} - \mu_{2,1}\mu_{3,2} + \mu_1\mu_{3,1}}{(\mu_1 - \mu_2)(\mu_1 - \mu_3)}\right)e^{-t\mu_1} \end{aligned} \tag{6.38}$$

$$p[0,t] = 1 - (p[3,t] + p[2,t] + p[1,t]) \tag{6.39}$$

Chapter 7

ADDITIONAL TOPICS

7.1 Traditional Bounds for Non-Binary Models

7.1.1 Introduction

Two issues complicate the task of bounding system expected values, as a function of component expected values, for non-binary reliability models:

1. For multistate and continuum components, knowing $E[X_i(t)]$ does not mean that one knows $F_i[x_i, t]$ (see Example 1).

2. Multistate and continuum coherent structures are not necessarily bounded by parallel and series arrangements of their components (see Example 2).

Example 1

Let us assume that $\phi(\mathbf{x}) = \min\{x_1, x_2\}$, $\Omega_i = \Omega = [0, 1]$. Let us also assume we know that $E[X_1] = E[X_2] = 1/2$. If the actual component distributions in one case are $P[X_i = 1/2] = 1$, which essentially means that each of the two components resides permanently in state 1/2, then we can calculate $E[\phi(\mathbf{X})] = 1/2$. If the actual component distributions are instead $P[X_i = 1] = 1/2$, $P[X_i = 0] = 1/2$, then we can calculate $E[\phi(\mathbf{X})] = 1/4$. Therefore, the expected value of the system is not unambiguously determined by the expected value of the components and the system's deterministic structure function.

Example 2

Consider the following continuum structure functions, which meet the coherence criteria of both Boedigheimer [214] and Baxter [195]:

$$\phi_1(\mathbf{x}) = \begin{cases} 1, & x_1 = x_2 = \cdots = x_n = 1 \\ \frac{\min\{x_1, x_2, \ldots, x_n\}}{10^{10}}, & \text{otherwise} \end{cases}$$

$$\phi_2(\mathbf{x}) = \begin{cases} 0, & x_1 = x_2 = \cdots = x_n = 0 \\ 1 - \frac{\min\{x_1, x_2, \ldots, x_n\}}{10^{10}}, & \text{otherwise} \end{cases}$$

For all absolutely continuous component distributions, $E[\phi_1(\mathbf{X})] \approx 0$ and $E[\phi_2(\mathbf{X})] \approx 1$. Therefore, one cannot always construct meaningful bounds based solely on "coherence" for non-binary models.

7.1.2 Trivial Bounds for Direct Selection

Some coherence definitions simply *assume* that the structure function will be bounded by certain functions (typically min and max), which then allows bounds to be constructed on any coherent system of that type based on the assumed structural extremes. Consider these five commonly-used structure functions: $\coprod_{i=1}^{n} x_i$, $\prod_{i=1}^{n} x_i$, $\min_{i=1}^{n} x_i$, $\max_{i=1}^{n} x_i$, $x_{(n-k+1)}$, and $\frac{\sum_{i=1}^{n} x_i}{n}$. Assume that the real system under consideration has independent subsystems and components, and that each subsystem (and any subsystems of it) is specified by one of these five structures.[1] Of all the possible systems which can be constructed under these assumptions, $\coprod_{i=1}^{n} x_i$ will have the highest state and $\prod_{i=1}^{n} x_i$ will have the lowest state, assuming that $\Omega = \Omega_i = [0, 1]$;

[1] n not necessarily equal for each, of course.

the expected value of these two may thus be used to bound $E[\phi(\mathbf{X})]$. In other words:

$$\prod_{i=1}^{n} x_i \leq \min_{i=1}^{n} x_i \leq \frac{\sum_{i=1}^{n} x_i}{n} \leq \max_{i=1}^{n} x_i \leq \coprod_{i=1}^{n} x_i \tag{7.1}$$

$$\min_{i=1}^{n} x_i \leq x_{(n-k+1)} \leq \max_{i=1}^{n} x_i \tag{7.2}$$

therefore

$$\prod_{i=1}^{n} E[X_i] \leq E[\phi(\mathbf{X})] \leq \coprod_{i=1}^{n} E[X_i] \tag{7.3}$$

7.1.3 Discrete Path-Cut Bounds with Partial Information

Lower Boundary Points Provided

Let us assume we are given a (possibly incomplete) set of lower boundary points for a discrete system. If a structure function is constructed based on these lower boundary points, a "lower bound" will result; the system state for any given component vector will be the same or lower as that which would be obtained if the set of lower boundary points provided were complete.[2] If only a lower bound is required, one could construct the structure function based on the provided lower boundary points, and then compute based on it any reliability measure one wishes; a measure such as $E[\phi(\mathbf{X})]$ constructed for this structure function will be a lower bound.

Forming valid upper and lower bounds on the probability of being above a certain state (without generating the structure function) is also possible for discrete models. The bounds given below are the result of taking the best set of bounds (lower and upper) for all "trivial bounds" given in (D.9); "forgetting" lower boundary points causes these bounds to become wider rather than narrower — as it should be.

$$\max_{\mathbf{y} \in L_k} \{\prod_{i=1}^{n} Q_{i,y_i}\} \leq Q_k \leq \min_{\mathbf{y} \in L_k} \{1 - \coprod_{i=1}^{n} (1 - Q_{i,y_i})\}$$

[2] If a component appears only as zero in all lower boundary points, it can be concluded either that this component is irrelevant or that some lower boundary points were omitted; if no lower boundary points are provided for a given system state (other than the minimal state), it should be investigated whether that system state may be removed from the model or merged with adjacent system states.

A similar expression can be computed for the upper boundary points.

Both Lower and Upper Boundary Points Provided

For the sake of simplicity we begin with the binary case. The ordinary path/cut bounds on $R \equiv Q_M$, based on the $R_i \equiv Q_{i,M}$ values, are:

$$\prod_{j=1}^{s}[1 - \prod_{i \in K_j}(1 - R_i)] \leq R \leq 1 - \prod_{j=1}^{r}(1 - \prod_{i \in P_j} R_i)$$

Defining $\beta_j^* = 1 - \prod_{i \in K_j}(1 - R_i)$ and $\alpha_j^* = 1 - \prod_{i \in P_j} R_i$, we note (since $0 \leq R_i \leq 1$ and $0 \leq 1 - R_i \leq 1$), that $0 \leq \beta_j^* \leq 1$ and $0 \leq \alpha_j^* \leq 1$. Rewriting the above expression in this new notation, we have $\beta_1^* \beta_2^* \cdots \beta_s^* \leq R \leq 1 - \alpha_1^* \alpha_2^* \cdots \alpha_r^*$. Because $0 \leq \beta_j^* \leq 1$ and $0 \leq \alpha_j^* \leq 1$, the lower bound may increase if a cut is omitted and the upper bound may decrease if a path is omitted. This is obviously not acceptable behavior if we wish for these bounds to be robust; it means that the more one forgets the tighter the bounds become.

The basic situation is that if one forgets a minimal path when constructing $\phi(\mathbf{x})$, there will be additional vectors such that $\phi(\mathbf{x}) = 0$ (where before those same vectors would have produced $\phi(\mathbf{x}) = 1$); therefore, one's calculated $\phi(\mathbf{x})$ will be the same or lower for any given $\mathbf{x}$. Similarly, $\phi(\mathbf{x})$ will be the same or higher if one forgets a minimal cut. This suggests that to have probability bounds remain valid in the face of incomplete path/cut information, paths should be used when computing lower bounds and cuts should be used when computing upper bounds.

This characteristic is possessed by the "min/max" bounds defined in Appendix D. Generalizing to the multistate case, we therefore conclude that equation (D.12) should be used to form probability bounds based on upper and lower boundary point sets which one suspects are incomplete.

7.2 Discretization

This section concerns determining the proper numbers and values of component and system states for a multistate model.

7.2.1 Customer-Interaction Approach

If the number of possible states for the component or system (in the real world, not the model) is finite, one need only determine if the customer requires the level of detail that would be present by explicitly including all of those states in the model. If not, discrete physical states may be grouped together into a smaller set of model states, according to the preferences of the customer [214].[3]

However, if the number of possible physical states is infinite, existing over a continuum of real numbers, one may either:

1. Utilize a continuum model.

2. Quantize these physical values so that they may be categorized into a finite number of model states.

If quantization is desired and interaction with the customer is possible, the following procedure may be used:

1. Start at the lowest possible physical state.

2. Increase the physical state until the customer indicates he or she wishes a state change for the model (i.e. that there is now a significant difference in his or her experience of the item in question which he or she wishes captured in the model for the purposes of effective decision-making).

---

[3]For systems or components which are *very* rarely in any state other than their optimal one, it may be necessary to distinguish only between the optimal state and the first sub-optimal state.

3. Any physical states that fall between this model state and the last model state are assigned to this new model state (see Figure 7.1 for a traditional multistate model example, where y represents the physical state and Y represents the model state).

4. If all possible physical states have been classified into a model state, stop. Otherwise return to step 2.

$$Y = \begin{cases} 0, & -\infty < y \leq y_1 \\ 1, & y_1 < y \leq y_2 \\ \vdots & \vdots \\ M, & y_m < y \leq \infty \end{cases} \tag{7.4}$$

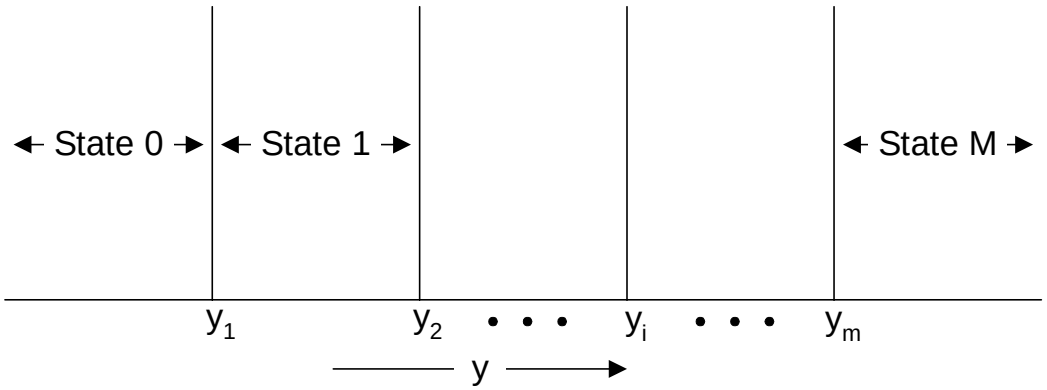


Figure 7.1: Traditional Discretization

If there are a finite number of economic countermeasures (i.e. repairs) which the customer may take to return the system or component to its optimal state, and if it is possible to quantify the economic loss caused by deviations from optimal, one could define the intermediary discrete model states as the physical states at which the customer would enact the available countermeasures.

7.2.2 Optimization Approach

The task is to take a continuous random variable which assumes values in $[\min, \max]$ and discretize it so that it only takes on values in $\{y_0, y_1, \ldots, y_m\}$, and to perform this discretization in a way that is, in some sense, "optimal." It is assumed that $y_0 = \min$ and that $y_m = \max$, so as to retain the natural extrema of the system (see Figure 7.2).

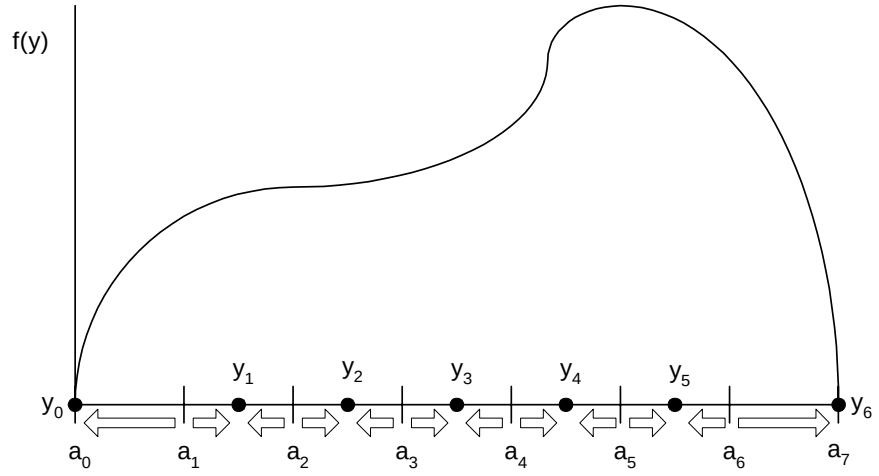


Figure 7.2: Discretization Pattern

The approach taken here will be to consider a discretization "optimal" (at least locally) if the RMS distance between a point in the original variable and its discretized value is minimized (at least locally). We assume knowledge of the PDF $f(y)$ for the original random variable. We define $d(y)$ as the point to which each y in the original variable will be discretized to,[4] and therefore write the minimization problem as follows: Minimize

$$g(a_1, a_2, \ldots, a_m)$$

[4]With regard to Figure 7.2, it is required that $a_0 = \min$, $a_{m+1} = \max$, and $a_i < a_j$ for all $i < j$.

where

$$[g(a_1, a_2, \ldots, a_m)]^2 = E[(y - d(y))^2] = \int_{\min}^{a_1} (y - \min)^2 f(y)\, dy + \int_{a_1}^{a_2} \left(y - \frac{a_1 + a_2}{2}\right)^2 f(y)\, dy + \cdots + \int_{a_i}^{a_{i+1}} \left(y - \frac{a_i + a_{i+1}}{2}\right)^2 f(y)\, dy + \cdots + \int_{a_{m-1}}^{a_m} \left(y - \frac{a_{m-1} + a_m}{2}\right)^2 f(y)\, dy + \int_{a_m}^{\max} (y - \max)^2 f(y)\, dy$$

subject to

$$\min < a_1 < a_2 < \cdots < a_m < \max$$

Please see Appendix I for illustrations of this optimization for various models.

7.3 Component and System Data Collection

When identical items are put on test to estimate their time-to-failure (in a binary sense), the items are generally monitored constantly so as to determine the exact failure times for each [66, 56]. Since the items are being constantly monitored, if one can discern and record finer gradations of performance than just "perfect functioning" and "complete failure" one need only record these exact performance levels continuously during testing to allow the construction of a non-binary, time-dynamic empirical distribution for that system or component's state [207].

Chapter 8

CASE STUDIES AND EXAMPLES

8.1 Misleading Discretizations

The examples in this section illustrate that it is possible for discretized models (multistate or binary) to lead to sub-optimal engineering decisions when the underlying system is inherently continuum. As many real-world systems are inherently continuum, this has real-world relevance.

8.1.1 Bicycle Example

Introduction

Let us assume that the system under consideration is a bicycle. It has two tires, each of which has a braking mechanism that slowly deteriorates as the system ages. We wish to examine the distance the bicycle requires in order to stop (from a standard speed) once the brakes are applied. We shall call the distance that the bicycle requires in order to stop when the brakes are in perfect condition system state 1, and the distance that the bicycle requires in order to stop when the brakes are in their worst condition system state 0. A binary model will allow only these two states for the system, a multistate model will allow a discrete set of states including 0 and 1, and a continuum model will allow the full continuum of states in $[0, 1]$.

We will examine three different models for this system, the aim being to illustrate how decision errors can be made as an inherently continuum system is discretized. Our approach will be to assume one structure function for all three of the systems; these three systems A, B, and C will be differentiated by having different distributions

for the states of their components.

The customer wishes to assess the reliability of the bicycle's brakes by examining the expected state of the system. Higher values are, of course, better. The "true" value for the expected state of each system will be obtained from the full continuum model. Under the assumption that, on occasion, only partial structure-function data is available, we will examine how our decisions would change when the following simplifications are used:

1. Scattered-Data Interpolation Continuum Model, 18 data points

2. Multistate Model, with states $\{0, 3/8, 5/8, 1\}$ for components and the system

3. Binary Model, with states $\{0, 1\}$ for components and the system

Structure Function

The following structure function (see Figure 8.1) is common to all the models considered in this bicycle example: $\phi(\mathbf{x}) = x_1 x_2$.

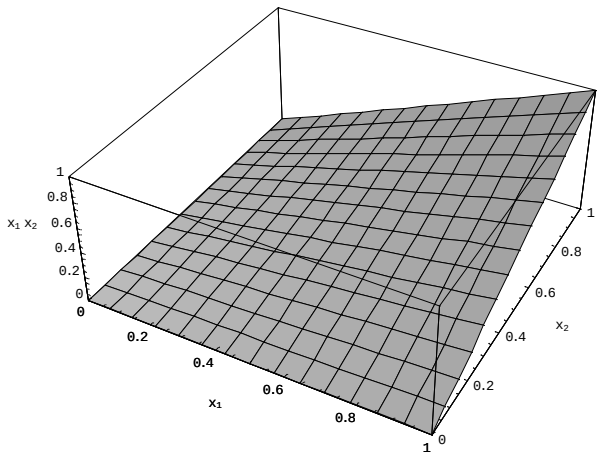


Figure 8.1: Bicycle System Underlying Structure Function

Component Distributions

Note: All of these component state distributions are s-truncated to have the domain $[0, 1]$.

System A

$X_1 \sim$ Normal(0.740818, 0.292007)

$X_2 \sim$ Normal(0.606531, 0.338651)

System B

$X_1 \sim$ Triangular(min=0, max=1, mode=4/5)

$X_2 \sim$ Triangular(min=0, max=1, mode=4/5)

System C

$X_1 \sim$ Beta(25/16, 1)

$X_2 \sim$ Beta(25/16, 1)

Interpolation Data

The data for the scattered-data model is presented in Table 8.1. The value of the adjustable parameter which was used is $\alpha^2 = 1/6$.

Discretization Patterns

For the multistate model with component states $\{0, 3/8, 5/8, 1\}$, component values in $[0, 1/4]$ are mapped to 0, values in $(1/4, 1/2]$ are mapped to $3/8$, values in $(1/2, 3/4]$ are mapped to $5/8$, and values in $(3/4, 1]$ are mapped to 1. The system state is determined by rounding the calculated value to the nearest value in the set $\{0, 3/8, 5/8, 1\}$.

For the binary model with component states $\{0, 1\}$, component state values in $[0, 1/2]$ are mapped to 0, and component state values in $(1/2, 1]$ are mapped to 1.

Table 8.1: Data for Bicycle System Scattered Data Interpolation

$\mathbf{x}$	$\phi(\mathbf{x})$
(0, 0}	0
(1, 1}	1
(0.4921976512515784, 0.6567022718937255)	0.3232273157976671
(0.4318405796090456, 0.916524025621182)	0.395792266449867
(0.1817978201605658, 0.01905256469642804)	0.00346371473027877
(0.0875176290939841, 0.4669830661251967)	0.0408692507743164
(0.5301873141694764, 0.822623711486269)	0.4361446561650311
(0.5282877576882771, 0.983386652896693)	0.5195111297993741
(0.0813181165792199, 0.4985288836434664)	0.04053942987822774
(0.05026429468732048, 0.5640697638801792)	0.02835256883588061
(0.2926663140648067, 0.3993713978357307)	0.1168825549474928
(0.2748510578765808, 0.5112077448767638)	0.1405059894740797
(0.1254410738042156, 0.1499798419835248)	0.01881363242739993
(0.949182608156626, 0.3951977675322207)	0.3751148477239091
(0.6332434225526374, 0.4932775700897993)	0.3123647767521129
(0.5173420285475804, 0.4786737419110386)	0.2476380446527176
(0.4514456023920716, 0.4742250053933712)	0.2140867932291938
(0.4298243994535963, 0.01169067578584187)	0.005024937698856182

The system state is determined by rounding the calculated value to the nearest value in the set $\{0, 1\}$.

For the sake of reference, the upper and lower boundary points for the multistate model are given in Tables 8.2 and 8.3.

Table 8.2: Bicycle System Upper Boundary Points

Point	Level
(0,1)	0
(1,0)	0
(3/8, 3/8)	0
(3/8, 1)	3/8
(5/8, 5/8)	3/8
(1, 3/8)	3/8
(5/8, 1)	5/8
(1, 5/8)	5/8

Table 8.3: Bicycle System Lower Boundary Points

Point	Level
(3/8, 5/8)	3/8
(5/8, 3/8)	3/8
(5/8, 1)	5/8
(1, 5/8)	5/8
(1, 1)	1

Results

Results are summarized in Table 8.4. Following each expected system state value in parentheses is the ranking this value gives that system. For example, based on the continuum ("true") model, we would recommend system C, followed by system A and system B.

Note that the scattered data interpolation model gives us the same ranking as

Table 8.4: Bicycle System Expected System States

	System A	System B	System C
Continuous	0.361647 (2)	0.360000 (3)	0.371802 (1)
Scattered-Data	0.368992 (2)	0.365522 (3)	0.380515 (1)
Multistate	0.404406 (3)	0.411163 (2)	0.411276 (1)
Binary	0.448679 (2)	0.472656 (1)	0.437498 (3)

the continuum model. The multistate model gives us the same first choice, but indicates incorrectly that system B would be our second choice rather than system A. The binary model gives us a completely incorrect ranking, indicating that system B is preferable to system C. This illustrates that using multistate, scattered data, or continuum modeling can result in better answers than binary simplification in some cases.

8.1.2 Automobile Example

Introduction

For this example, we assume that the system in question (the automobile) is inherently continuum, and has exactly four components (its tires). The four components may take on values in $[0, 1]$, as may the system. The state of each component (and hence the system) is stochastic, and is given by a PDF. Although the distributions for the four components are not necessarily identical, they are assumed to be independent.

Our approach shall be to consider four different systems, and examine how these four distinct systems are ranked when the full continuum model ("reality") is used, when a multistate model is used, and when a binary model is used. The ranking of the systems shall be determined by decreasing order of the expected value of the structure function for that system.

The Discretization Used

For the binary model, the components are discretized by considering all states in $(1/2, 1]$ as being mapped to state 1, and all states in $[0, 1/2]$ as being mapped to state 0. The appropriate probabilities are calculated from the PDF for each component. System states are mapped to $\{0, 1\}$ in the same fashion.

For the multistate model, the situation is identical except that for each component the states in $[0, 1/8]$ are mapped to state 0, the states in $(1/8, 3/8]$ are mapped to state $1/4$, the states in $(3/8, 5/8]$ are mapped to state $1/2$, the states in $(5/8, 7/8]$ are mapped to state $3/4$, and the states in $(7/8, 1]$ are mapped to state 1.

The Systems Considered

Note: All distributions are s-truncated (if necessary) so that $\int_0^1 f_i(x_i)\, dx_i = 1$.

System D

$\phi(\mathbf{x}) = x_1 x_2 x_3 x_4$

$X_1, X_2, X_3, X_4 \sim$ Beta[25/16,1]

System E

$\phi(\mathbf{x}) = (\ln(x_1 + 1)/\ln(2))((e^{x_2} - 1)/(e - 1))(x_3^4)(\sqrt{x_4})$

$X_1, X_2, X_3, X_4 \sim$ Normal[3/4,1/5]

System F

$\phi(\mathbf{x}) = (\ln(x_1 + 1)/\ln(2))/4 + ((e^{x_2} - 1)/(e - 1))/4 + (x_3^{10})/4 + (\sqrt{x_4})/4$

$X_1, X_2, X_4 \sim$ Normal[1/10,1/10]

$f_3[x_3] = 2(1 - x_3)$

System G

$\phi(\mathbf{x}) = x_1^2 x_2^4 x_3^{1/2} x_4^{1/8}$

$X_1 \sim$ Weibull[3/2,2]

$X_2 \sim$ Weibull[3,4]

$X_3 \sim$ Weibull[2,5]

$X_4 \sim$ Weibull[6,3]

Results

The expected state for each system and model type is given in Table 8.5. The systems are therefore ranked (in decreasing order of preference) as given in Table 8.6.

Table 8.5: Automobile Expected State of the Systems

	System D	System E	System F	System G
Continuum	0.138	0.132	0.152	0.133
Multistate	0.122	0.130	0.119	0.145
Binary	0.191	0.605	0.000	0.389

Table 8.6: Automobile System Rankings

Continuum	F, D, G, E
Multistate	G, E, D, F
Binary	E, G, D, F

We therefore note that different rankings are produced depending on whether the model is discretized or not. In reality (continuum model) system F is the best, but the multistate approximation would instead recommend system G to the customer, and the binary approximation would instead recommend system E to the customer.

8.2 Time-Dynamic Models

8.2.1 Tire Example

Let us consider a simple non-repairable system with three states: $\Omega = \{0, 1, 2\}$ (see Table 8.7). The system begins in its best possible state (2), where it spends a length of time given by an exponential distribution with transition rate μ_2. After leaving state 2, the system spends a length of time in state 1 given by an exponential distribution with transition rate μ_1. After leaving state 1 it enters state 0 (the worst possible state) where it remains. We assume $\mu_2 \neq \mu_1$, $\mu_2 > 0$, and $\mu_1 > 0$.

Table 8.7: Tire Example States

State	Tire Condition
2	Low Tread Wear
1	Moderate Tread Wear
0	High Tread Wear

System state probabilities are given as follows:

$$\begin{cases} p[2,t] = e^{-\mu_2 t} \\ p[1,t] = \frac{\mu_2}{\mu_2 - \mu_1}(e^{-\mu_1 t} - e^{-\mu_2 t}) \\ p[0,t] = 1 + \frac{\mu_2}{\mu_1 - \mu_2}e^{-\mu_1 t} - \frac{\mu_1}{\mu_1 - \mu_2}e^{-\mu_2 t} \end{cases} \tag{8.1}$$

The expected value of $\phi(\mathbf{X})$, the system state (as a function of t), is therefore

$$E[\phi(\mathbf{X})] = \frac{2\mu_1 - \mu_2}{\mu_1 - \mu_2}e^{-\mu_2 t} - \frac{\mu_2}{\mu_1 - \mu_2}e^{-\mu_1 t} \tag{8.2}$$

Let us assume by way of example that we have the following parameters for the tire example: $\mu_2 = 2$, $\mu_1 = 1$. For this special case, we have $p[2,t] = e^{-2t}$, $p[1,t] = 2(e^{-t} - e^{-2t})$, $p[0,t] = 1 + e^{-2t} - 2e^{-t}$, and $E[\phi(\mathbf{X})] = 2e^{-t}$.

8.2.2 Military Example

Model Definition

This model is based upon an example originally presented by Boedigheimer and Kapur [146], and considers the effectiveness of a military force (the system) which begins battle with six attack units and five artillery units (the components). The components x_1 and x_2 are defined as the remaining number of units of their type: a component state vector of $(3, 4)$ would indicate there are three remaining attack units and four remaining artillery units. We assume it is not possible to replace units that are destroyed, and that each component x_i spends a length of time in each of its non-minimal states governed by an exponential distribution with a common parameter μ_i (where $i \in C$).

We compute in (8.3) and illustrate in Figure 8.2 the dynamic probabilities of each component being in each one of its states as a function of time:

$$p_i[x_{ij}, t] = \begin{cases} \frac{(\mu_i t)^{m-j} e^{-\mu_i t}}{(m-j)!} & \text{for } j = m, m-1, \ldots, 1 \\ 1 - \sum_{j=1}^{m} p_i[x_{ij}, t] & \text{for } j = 0 \end{cases} \tag{8.3}$$

Our customer (the military) indicates they wish to consider six distinct levels of performance for the entire military force: $\{0, 1, 2, 3, 4, 5\}$. It is assumed that the state values for the system have some direct interpretation in terms of the real performance of the system (e.g. state 2 is half as desirable as state 4). Based on their experience with this type of force, the military indicates that the component state vectors given in Table 8.8 are such that any decrease in the state of either component would cause a decrease in the system state. Hence, they are the lower boundary points.

From this information, using commonly known techniques for multistate systems [144] we can determine the system state associated with any particular state of the set of components. If the model were large enough and complicated enough that computational time was a concern, then it might be worthwhile to decompose the

Table 8.8: Customer-Specified Lower Boundary Points

LBP	LEVEL
(1,2)	1
(2,0)	1
(1,2)	2
(2,1)	2
(1,2)	3
(2,3)	4
(4,4)	5
(5,3)	5

system into a series of binary models [135] and then utilize some of the more advanced algorithms which have recently been developed to assist in such cases [36, 52].

However, let us assume that through historical battle records or other simulations it is found that the transition rate for the first component is $\mu_1 = 1.2000$, and the transition rate associated with the second component is $\mu_2 = 1.9000$. With this information, using direct enumeration based on the structure function calculated from the lower boundary points, we may find the probabilities of the system being in any given state as a function of time.

With this time-based information, we may calculate values for each of the reliability measures of interest. Measures will be calculated as functions of time and specific results will be returned for $t = 2$, which is assumed to be of special interest to the customer.

Performance at the Customer's Time of Interest ($t = 2$)

Please refer to Table 8.9 and 8.10.

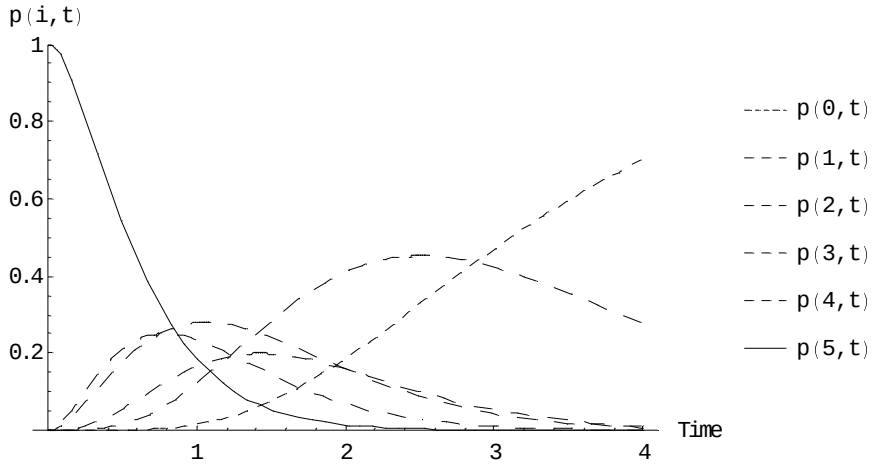


Figure 8.2: Dynamic System State Probabilities

Table 8.9: State Probabilities at $t = 2$

i	$p[i, 2]$
0	0.1876
1	0.4100
2	0.1593
3	0.1595
4	0.0690
5	0.0146

Table 8.10: Selected Reliability Measures

$E[\phi(2)]$	$E[\phi(2)]/M$	$V[\phi(2)]$	TOE	OE[2]	OVUB[2]
1.5562	0.3112	1.5303	8.4583	6.4648	6.5551

Interpretation of Measures at the Customer's Time of Interest ($t = 2$)

We can conclude that the system is in a state of disrepair, as the expected state is only 31% of maximum. It can also be instructive to examine the ratio of OE[τ] and

TOE, which represents the expected fraction of the total benefits from the system which the customer has already received. At $t = 2$, this ratio is 76%.

One may examine OVUB[τ] with Chebychev's inequality [274] to get a better sense of the behavior of OE[τ]. Of course, as OVUB[τ] is already an upper bound for the variance of the quantity under consideration, this bound will be quite conservative. For example:

$$P[1.380 < \int_0^\tau \phi(t)\, dt \leq 11.550] \geq \frac{3}{4} \tag{8.4}$$

Assessment of System Dynamic Performance

We start by examining $E[\phi(t)]$, the average state of the system as a function of time (see Figure 8.3).

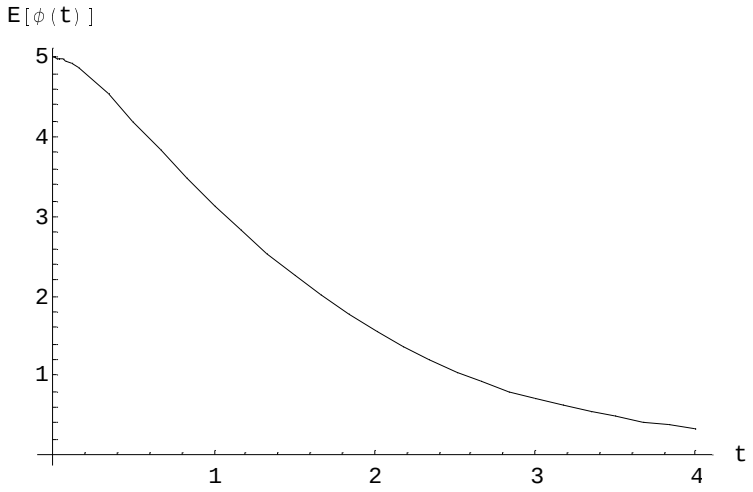


Figure 8.3: System State Expectation

We now examine the integral of the average state of the system, OE[τ] (see Figure 8.4). OE[τ] asymptotically approaches TOE, which is 8.4583 in this example; TOE represents the expected total utility one will receive from the system. As this is a non-repairable system, we would expect the average length of time (DTE[i]) the

system spends above each state[1] to be positive and finite (see Table 8.11).

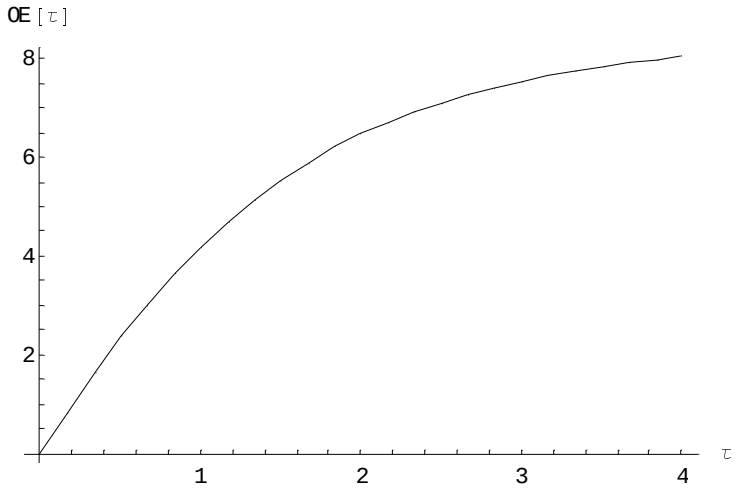


Figure 8.4: System Output Expectation

Table 8.11: Values of DTE[i]

DTE[0]	DTE[1]	DTE[2]	DTE[3]	DTE[4]	DTE[5]
3.4036	1.8981	1.5114	1.0000	0.6452	0.0000

We now examine $V[\phi(t)]$, and note that it has a maximum at $t = 1.37$. One can easily imagine situations where the variance of the state of the system might be of equal or greater interest to the customer than the expected state of the system.

The LWE measure was calculated for six different $U(t)$ functions (see Figures 8.6, 8.7, 8.8, 8.9 and Table 8.12).

$U(t)$ functions #1 and #2 may accurately model the time-varying nature of the customer's interest in the product in cases where the product or service is of greatest value when it is initially purchased, but where there is a steadily decreasing positive

[1] other than the maximal state

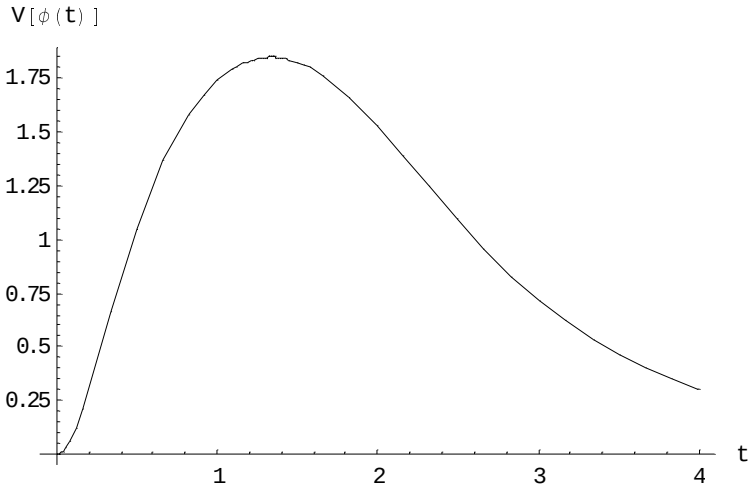


Figure 8.5: System State Variance

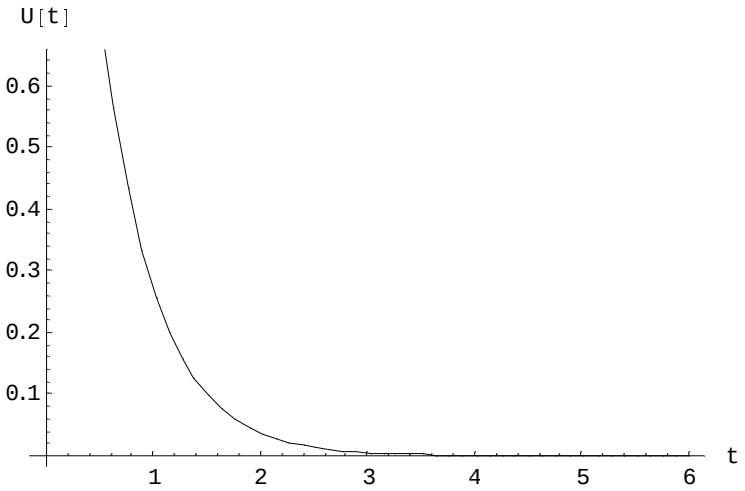


Figure 8.6: $U(t)$ Function #1

utility for all $t > 0$ (perhaps being especially true in the low-cost consumer electronics industry). For our military example, this could be reflected in a situation where a city population is being defended while it evacuates.

$U(t)$ functions #3 and #4 reflect cases where customer interest grows to a maximum (at $t = 3$ for #3 and at $t = \sqrt{1/2}$ for #4) and then steadily fades. For our military example, this general form of $U(t)$ could be appropriate when an impor-

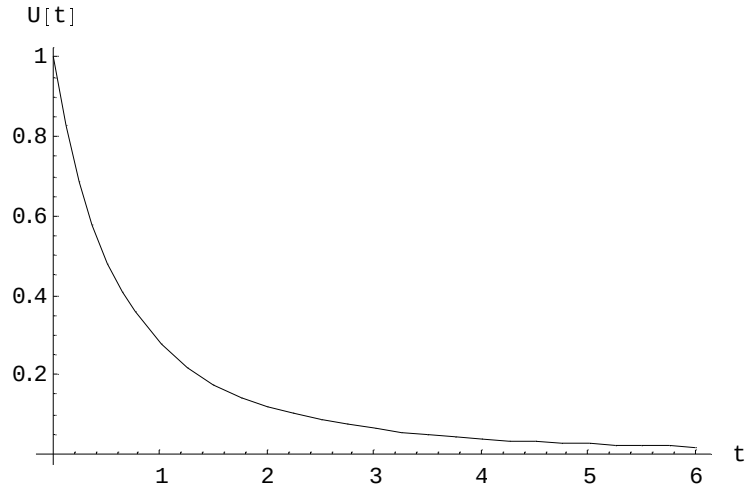


Figure 8.7: $U(t)$ Function #2

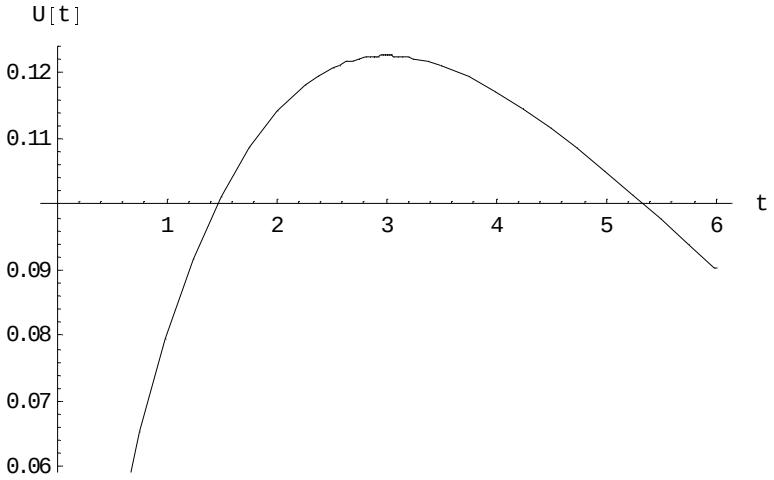


Figure 8.8: $U(t)$ Function #3

tant offensive is being planned at some unspecified time in the future, but when the military force in question must fight until the time of that offensive.

The $U(t)$ functions #1 through #4 are continuous probability density functions (PDF's): #1 is an Exponential PDF, #2 is an FRatio PDF, #3 is a Gamma PDF, and #4 is a Weibull PDF. Because these $U(t)$ functions are PDF's (presumably for the time-of-occurrence of some future event), the LWE measures could be interpreted

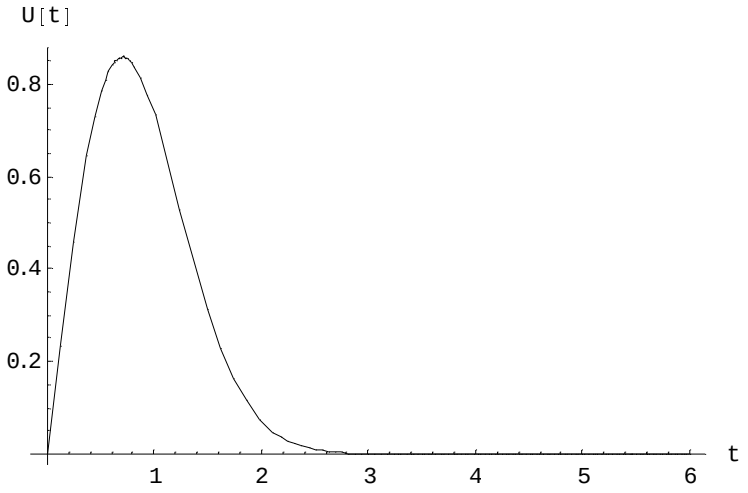


Figure 8.9: $U(t)$ Function #4

Table 8.12: $U(t)$ Definitions and LWE Values

#	$U(t)$	LWE
1	$2e^{-2t}$	4.1944
2	$9\sqrt{3}(3+2t)^{-5/2}$	2.9228
3	$(t/9)e^{-t/3}$	0.62776
4	$2te^{-t^2}$	3.4351

as the expected state of the system at the time of this event. As one can see, and as one would expect, $U(t)$ functions which are large for small t and small for large t produce large ("good") values for LWE.

8.2.3 Continuum Example

In this subsection we illustrate the use of Tables 3.1 and 3.2. Each of the two systems in this example will use the same six components in different structural arrangements. The six components have distributions as given in Table 8.13.

CN denotes a cumulative normal distribution: $\mathrm{CN}(s;\mu,\sigma) = \int_{-\infty}^{s} \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}}\, dx$. CNT denotes a cumulative normal distribution s-truncated to lie between 0 and 1: $\mathrm{CNT}(s;\mu,\sigma) = \frac{\mathrm{CN}(s;\mu,\sigma)-\mathrm{CN}(0;\mu,\sigma)}{\mathrm{CN}(1;\mu,\sigma)-\mathrm{CN}(0;\mu,\sigma)}$. As we wish to make these distributions functions of time in modeling a non-repairable system, it is reasonable to assume that μ will decrease over time. We additionally assume that σ will increase to a maximum when $\mu = \frac{1}{2}$ and then gradually decrease. We therefore define, for the sake of these examples, $\mu(\epsilon,t) = e^{-\frac{\log(2)}{\epsilon}t}$ and $\sigma(\epsilon,t) = \frac{1}{4}\left(\frac{1}{2} - \left|\frac{1}{2} - \mu(\epsilon,t)\right|\right) + \frac{1}{100}$. Note that the parameter ϵ is inversely proportional to the rate of decay of that component's expected state.

Table 8.13: Continuum Example Component Distributions

i	Distribution of X_i
1	Multistate, with $p[\phi_j,t]$ given by Table 2.2 with $\lambda = 3$
2	Binary, with $p(t) = e^{-\frac{3t}{2}}$
3	Continuum, with $F[s,t] = \mathrm{CNT}(s;\mu(2/3,t),\sigma(2/3,t))$
4	Continuum, with $F[s,t] = \mathrm{CNT}(s;\mu(1,t),\sigma(1,t))$
5	Continuum, with $F[s,t] = \mathrm{CNT}(s;\mu(1/2,t),\sigma(1/2,t))$
6	Continuum, with $F[s,t] = \mathrm{CNT}(s;\mu(3/4,t),\sigma(3/4,t))$

System A

In this example we wish to demonstrate the calculation of time-dynamic reliability measures for a system of six components arranged in "minimum" and "maximum" subsystems as illustrated in Fig. 8.10: $\phi(\mathbf{X}) = \max(X_2, \min(X_1, \max(X_3, X_4, X_5, X_6)))$. Using the expressions given in section 2.2 and Table 3.1, we can compute the measures given in Table 8.14 along with those illustrated in Fig. 8.11, Fig. 8.12, and Fig. 8.13.

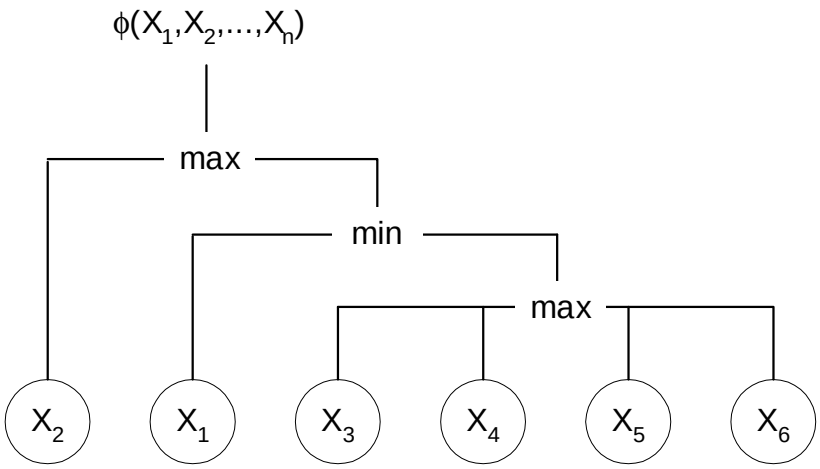


Figure 8.10: System A Structure Function

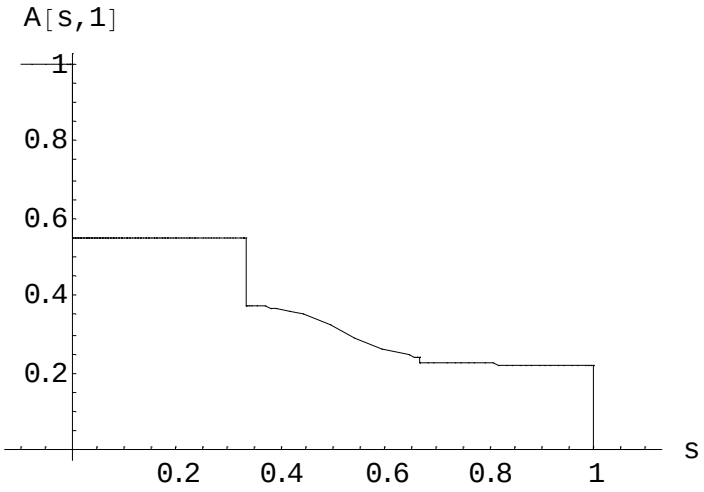


Figure 8.11: System A Availability (at $t = 1$)

System B

In this example we wish to demonstrate the calculation of reliability measures at one moment in time ($t = 1$) for a system of the same six components arranged in "average," "product," and "coproduct" subsystems (rows 4, 5, and 6 respectively of Table 3.2), identified as $\sum$, $\prod$, and $\coprod$ in Fig. 8.14: $\phi(\mathbf{X}) = 1 - (1 - X_6)(1 - ((X_1 +$

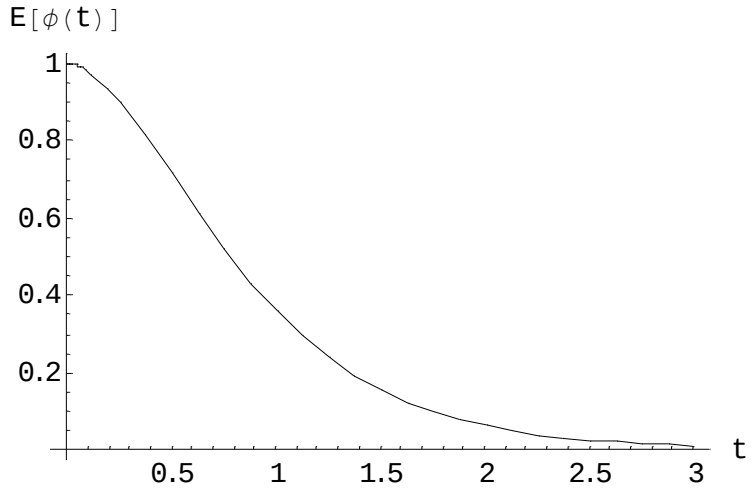


Figure 8.12: System A State Expectation

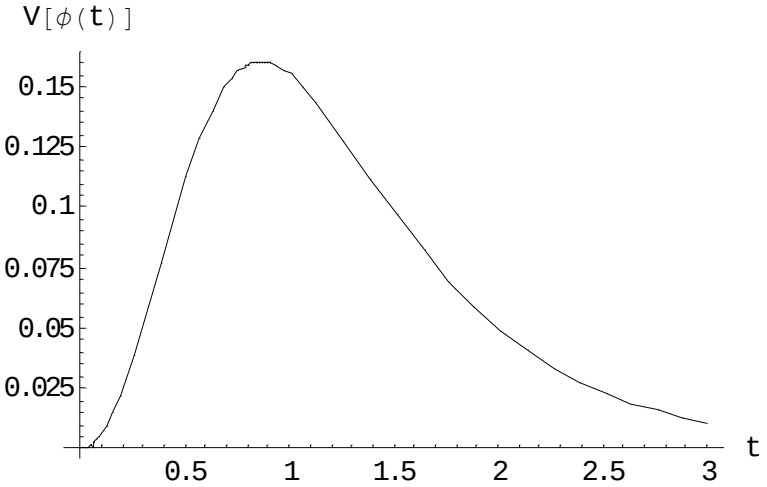


Figure 8.13: System A State Variance

$X_2 + X_3)/3)X_4X_5)$.

Using Table 3.2, we may compute $E[\phi(1)] = 0.417$ and $E[\phi^2(1)] = 0.185$ for this system with component distributions as given in Table 8.13. This allows us to compute $V[\phi(1)] = E[\phi^2(1)] - E[\phi(1)]^2 = 0.011$. By computing these values for other values of t, any of the measures discussed in this dissertation which are functions of $E[\phi(t)]$ and $E[\phi^2(t)]$ may also be computed.

Table 8.14: System A Measures

Measure	Value
OE[3]	0.918
OVUB[3]	0.874
TOE	0.926
DTE[0]	1.198
DTV[0]	0.426
$B[0, 0.05]$	0.389
$T[0.95]$	1.890

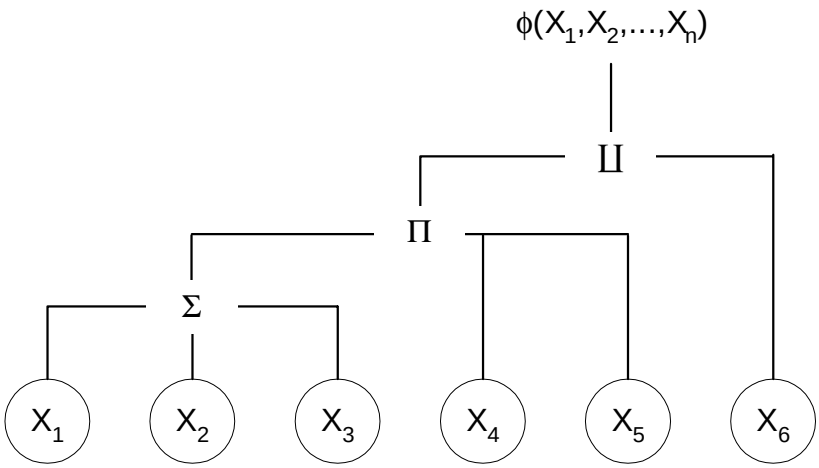


Figure 8.14: System B Structure Function

8.3 Large Static System

This mixed system is composed of "parallel" ($\phi(\mathbf{x}) = \max \mathbf{x}$) and "series" ($\phi(\mathbf{x}) = \min \mathbf{x}$) subsystems. The reliability block diagram is given in Figure 8.15, and the

distributions for each component are given in Table 8.15.[2]

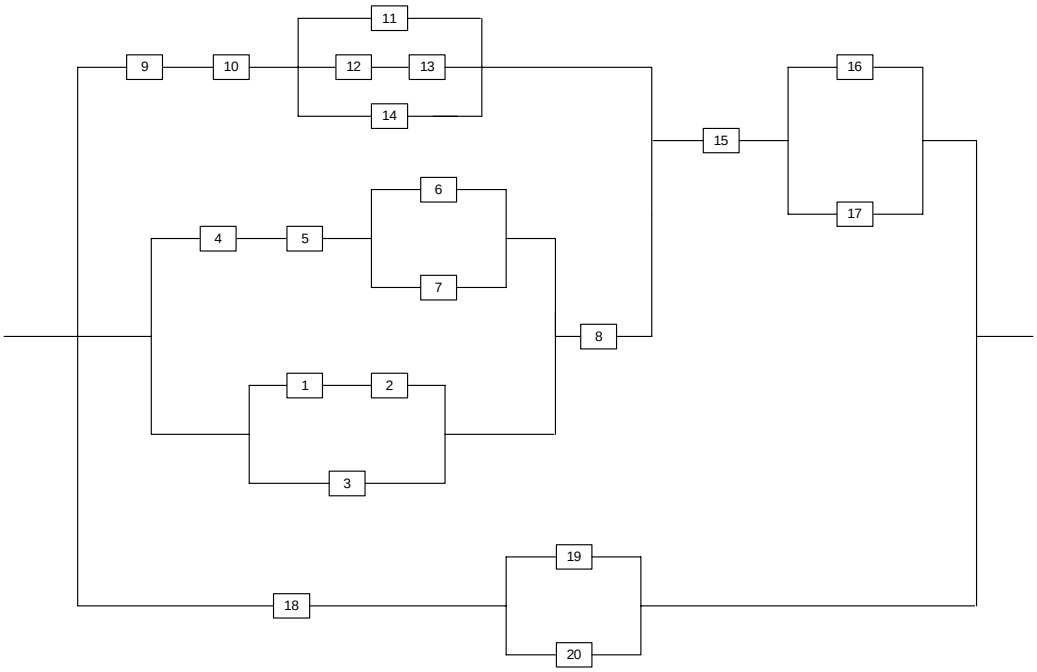


Figure 8.15: "Large Static System" System Diagram

"Binary" is a discrete distribution whose probability mass function is

$$P[X = x] = \begin{cases} p, & x = 1 \\ 1 - p, & x = 0 \\ 0, & \text{otherwise} \end{cases} \tag{8.5}$$

"UniformMixed" is a mixed continuous/discrete distribution whose CDF is

$$F[x] = \begin{cases} 0, & x < 0 \\ 1, & x \geq 1 \\ p0 + x(1 - p0 - p1), & \text{otherwise} \end{cases} \tag{8.6}$$

[2]The definitions for these distributions are given in Appendix H (all continuous distributions are h-truncated) and in Equations 8.5 and 8.6.

Table 8.15: "Large Static System" Component Distributions

Component	Distribution
1	Binary, p=0.5
2	UniformMixed, p0=0.2, p1=0.3
3	Chi(1)
4	Binary, p=0.432
5	Binary, p=0.643
6	UniformMixed, p0=0.05, p1=0.48
7	Normal(0.52, 0.2)
8	Beta(0.7, 0.31)
9	Cauchy(0.3, 2)
10	ChiSquare(1)
11	Exponential(2)
12	ExtremeValue(0.6,3)
13	FRatio(5,7)
14	Gamma(0.8, 0.3)
15	Laplace(0.2, 0.238)
16	LogNormal(0.2,1)
17	Logistic(0.222, 0.6)
18	Rayleigh(1)
19	StudentT(4)
20	Weibull(0.4, 0.9)

Using Table 3.1, the CDF of the system state may be computed as shown in Figure 8.16 and the expected value of the system state is $E[\phi(\mathbf{x})] = 0.61$.

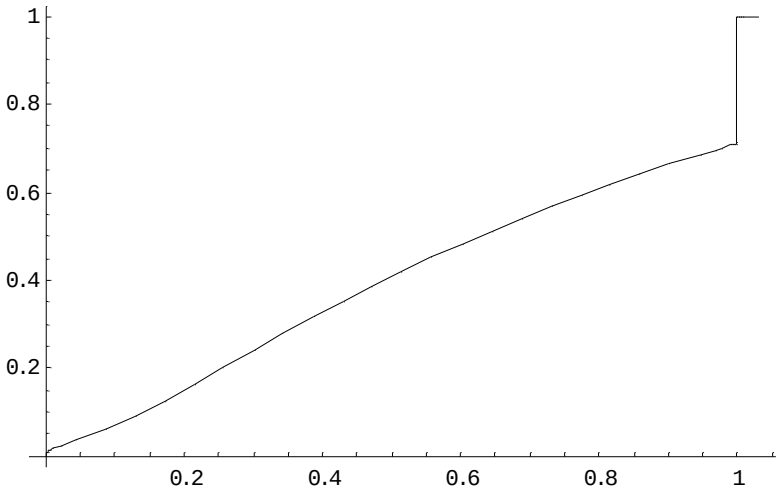


Figure 8.16: Large Static System CDF

8.4 Aviation Communications

This example consists of a multistate reliability model which was proposed and used during the author's research appointment at AT&T Wireless Communications during 1995. Data has been slightly altered to protect confidentiality.

An airplane's air-to-ground telephony system has two communications channels. When both channels are open and functioning properly, the channel to be used when an outgoing call is placed is selected randomly by the telephony software. Therefore, when the system is operating perfectly the two channels should be chosen in approximately equal proportion (available data confirmed this was true to within 0.1%). Deviations from this (over and above what would be expected by chance, based on the geometric distribution) indicate that one channel is not functioning and hence not accepting calls; this is a highly undesirable situation, as it requires customers to wait longer for an open channel and increases the probability that some may balk from the queue. Therefore, the system's best state occurs when approximately 50% of the calls are being made on Channel 1. To enhance understanding of this system a discrete time Markov chain, where state changes are said to occur every 30 calls, was

created.

Note that this analysis cannot detect situations where both channels are failing. From the call records, this is indistinguishable from the plane simply not flying that day, and hence not allowing any calls. It was also the opinion of the engineers responsible for this system that situations where both channels fail simultaneously are quite rare and could safely be ignored.

The states for this system are enumerated in Table 8.16. Note that each of the two channels is in either poor (0), moderate (1), or good (2) condition. As was indicated earlier, this model (and the data it is based on) cannot detect uniform changes in the functionality of both channels simultaneously, so states (1,1) and (0,0) are essentially conflated with (2,2). Also, (0,1) is conflated with (1,2) and (1,0) is conflated with (2,1). However, if the engineers are correct in assuming that both channels very rarely degrade simultaneously, then we may consider these states as providing separate information on independent components (see Figure 8.17). As each component (channel) may be in more than two states, this is a two-component multistate reliability model.

Table 8.16: Aviation Communications State Definitions

SYSTEM STATE	DESCRIPTION	CH1	CH2
0	0%-20% of calls on CH1	0	2
1	80%-100% of calls on CH1	2	0
2	20%-40% of calls on CH1	1	2
3	60%-80% of calls on CH1	2	1
4	40%-60% of calls on CH1	2	2

As Table 8.17 reveals, when the system is either in one of its two poorest states (0 or 1) or else in its best state (4), it is likely to stay there; if it is between these two

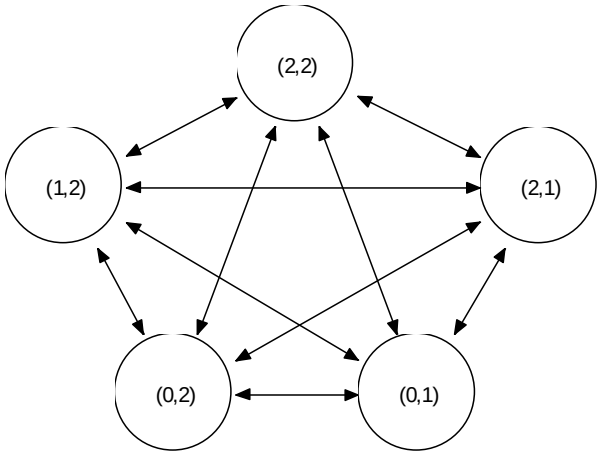


Figure 8.17: Aviation Communications Markov Chain

Table 8.17: Aviation Communications Transition Probabilities

States	0	1	2	3	4
0	0.35	0.00	0.20	0.05	0.40
1	0.00	0.50	0.05	0.20	0.25
2	0.00	0.00	0.10	0.15	0.75
3	0.00	0.00	0.10	0.15	0.75
4	0.00	0.00	0.10	0.15	0.75

extremes (2 or 3), it is likely to improve in the next time step.[3] The system tends to improve to perfect functionality at the next transition if one of the channels is only in a moderate state of failure rather than a severe state of failure.

[3]In other words, the system tends to remain as it is either if one channel is completely failed or else if both channels are operating perfectly.

8.5 Quadratic Loss Function Examples

8.5.1 Introduction

Following the work of Genichi Taguchi, much of the modern quality literature shifts the focus of quality engineering from "meeting specifications" to reducing variation around the target value for the quality characteristic under consideration [253]; one important quality characteristic for a pencil, for example, might be the hardness of the lead, which can be quantified for each pencil and which can be brought on average closer to target through application of robust design techniques. If variation from the target value for a quality characteristic of a product or service produces a loss to the customer, then it is valuable to quantify the extent of this loss so that improvements may be assessed.

Existing models for quality loss using quadratic loss functions, when extended into the time domain, lend themselves readily to modeling using standard continuum and multistate reliability models. The quadratic loss function is treated in this section as a type of reliability measure, and its expected value, variance, skewness, and kurtosis are found in terms of easily calculable quantities.

If y_0 is the desired ("target") value of a quality characteristic and y is the value it assumes for a given product, the quadratic quality loss function is written as $L(y) = k(y - y_0)^2$. Clearly, the quality loss $L(y)$ is zero when the target y_0 is met. The quality loss coefficient, k, is calculated for "nominal is best" quality characteristics as $k = A_0/\Delta_0^2$, where A_0 is the cost to the customer incurred by a deviation of Δ_0 from the target value y_0 for some specific value of Δ_0 (often calculated through examination of the costs for repairs which bring the quality characteristic y back to target[4]). This loss function is derived by taking the Taylor series expansion around a local minimum

[4]For the time-dynamic model presented here, A_0 will be cost-per-unit-time rather than cost (as it is in the time-invariant case); of course, once the expected cost-per-unit-time is found ($E[L[\phi(t)]]$), it may be integrated over a period of time to find the expected total cost that will be incurred in that period of time due to deviations from target.

of a general loss curve, and dropping terms of order higher than 2.

In this section, we will demonstrate an approach to this model which allows y to be a function of time, so that the time-based performance of this quality characteristic may add additional insight into the nature of the variation-induced losses to the customer. When y is allowed to be a function of time, it will be shown that the resulting model may be fruitfully treated as a continuum and/or multistate reliability model, with the quadratic loss function as a specific type of reliability measure.

Although it is not common in the reliability literature to consider cases where the optimal value of a random variable is anything other than its maximal value, in the quality literature it is quite common. For example, if there is a cost associated with overproduction, and the quality characteristic under consideration is the production rate of a refinery, then there will certainly be a cost associated with upward deviations from nominal. Such cases will be considered in this section.

The use of quadratic loss functions in reliability measures was hinted at in a recent doctoral dissertation example [214], but has not been developed in the literature. The concept of reliability models where the maximal state might not be optimal (due to varying customer demands) was originally explored by Aven [188].

8.5.2 Methodology

In essence, we wish to find the expected value, variance, skewness, and kurtosis of a function $k(\phi(t) - y_0)^2$ of a time-dynamic random variable $\phi(t)$, and calculate all of these reliability measures as functions of central moments of $\phi(t)$. We would like to be able to write the expressions for the expected value, variance, skewness, and kurtosis of the quadratic loss function of $\phi(t)$ as a function of the CDF for $\phi(t)$, as density functions are not available for the continuous component of mixed random variables unless the discrete and continuous elements of its distribution are separated. As it is more convenient not to have to separate the discrete and continuous components of the CDF of $\phi(t)$ for all values of t, we shall proceed with an approach which will

utilize only $F[x,t]$ in its original form.

Employing a theorem proved in [263] for general CDF's (and which has been used earlier in this dissertation), we introduce the following expressions for the central moments of random variables:

$$m_i = E[X^i] = \int_0^\infty i x^{i-1}[1 - F_X(x) + (-1)^i F_X(-x)]\,dx$$

of course, if X is a non-negative random variable, this reduces to

$$m_i = E[X^i] = \int_0^\infty i x^{i-1}[1 - F_X(x)]\,dx$$

Note that although the integrand in the above expressions may have up to a countably infinite number of discontinuities (if the model has a discrete component), it is still a standard Riemann integral, and many software packages have been written which will efficiently perform quadrature on it.

Utilizing this notation, we may write the desired expressions for the mean, variance, skewness, and kurtosis of the quadratic loss function of a random variable in terms of the m_i terms (after some algebra) as follows:

$E[L[\phi(t)]]$

$$k(y_0^2 - 2y_0 m_1 + m_2)$$

$V[L[\phi(t)]]$

$$k^2(-4y_0^2 m_1^2 + 4y_0^2 m_2 + 4y_0 m_1 m_2 - m_2^2 - 4y_0 m_3 + m_4)$$

Skewness$[L[\phi(t)]]$

$$\frac{\begin{array}{l}2(y_0^2 - 2y_0 m_1 + m_2)^3 \\ -3(y_0^2 - 2y_0 m_1 + m_2)(y_0^4 - 4y_0^3 m_1 + 6y_0^2 m_2 - 4y_0 m_3 + m_4) \\ +y_0^6 - 6y_0^5 m_1 + 15y_0^4 m_2 - 20y_0^3 m_3 + 15y_0^2 m_4 - 6y_0 m_5 + m_6\end{array}}{(-4y_0^2 m_1^2 + 4y_0^2 m_2 + 4y_0 m_1 m_2 - m_2^2 - 4y_0 m_3 + m_4)^{3/2}}$$

Kurtosis$[L[\phi(t)]]$

$$\frac{\begin{array}{l}-3(y_0^2-2y_0m_1+m_2)^4\\+6(y_0^2-2y_0m_1+m_2)^2(y_0^4-4y_0^3m_1+6y_0^2m_2-4y_0m_3+m_4)\\-4(y_0^2-2y_0m_1+m_2)(y_0^6-6y_0^5m_1+15y_0^4m_2-20y_0^3m_3+15y_0^2m_4-6y_0m_5+m_6)\\+y_0^8-8y_0^7m_1+28y_0^6m_2-56y_0^5m_3+70y_0^4m_4-56y_0^3m_5+28y_0^2m_6-8y_0m_7+m_8\end{array}}{(-4y_0^2m_1^2+4y_0^2m_2+4y_0m_1m_2-m_2^2-4y_0m_3+m_4)^2}$$

8.5.3 Example 1

Let us first consider an example where the optimal value is the maximal value for the system: $L[\phi(t)] = (1/2)(\phi(t)-1)^2$ (see Figure 8.18).

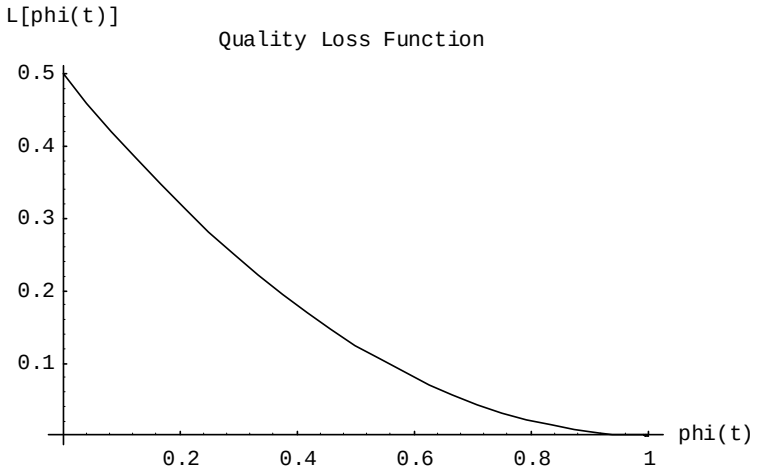


Figure 8.18: Example 1 Quality Loss Function

We consider a random variable for $\phi(t)$ which is normally distributed, h-truncated between 0 and 1, with mean 5/6 and standard deviation 1/2 (see Figure 8.19), and compute the various functions of the quadratic loss function (see Table 8.18).

Of course, it should be emphasized that the loss function obtained is the loss per unit time. Thus, if the CDF for $\phi(t)$ and the values of k and y_0 are time invariant the mean quality loss (in units of cost) over some time period Δt would be $E[L[\phi(t)]]\Delta t$.

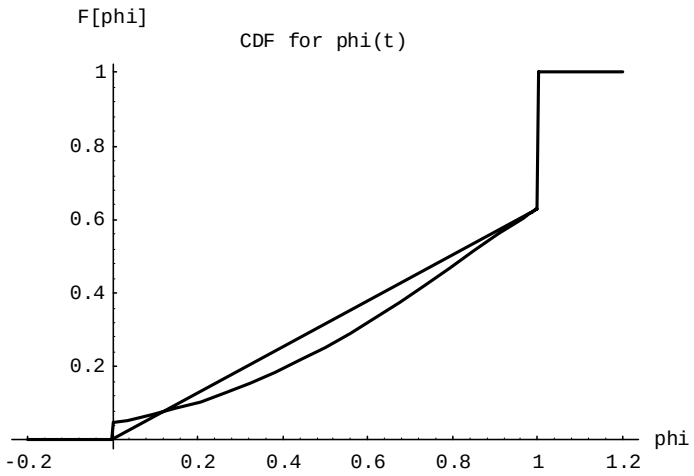


Figure 8.19: Example 1 Distribution

Table 8.18: Example 1 Measures

$E[L[\phi(t)]]$	0.0915454
$V[L[\phi(t)]]$	0.0202867
Skewness$[L[\phi(t)]]$	1.71532
Kurtosis$[L[\phi(t)]]$	4.86313

If these expressions were not time invariant we could calculate the mean cost due to variation as $E[\int_{t_0}^{t_0+\Delta t} L[\phi(t)]\,dt] = \int_{t_0}^{t_0+\Delta t} E[L[\phi(t)]]\,dt$

8.5.4 Example 2

Let us now consider an example where the optimal value is not the maximal value for the system. We define the random variable $\phi(t)$ to be the product output rate of a factory, where demand for its product (per unit time) is y_0. If the factory produces at a rate greater than the demand, the excess production must be moved *into* storage, at a cost per unit time proportional to the overproduction per unit time. If the factory produces at a rate less than the demand, the product must be moved *from* storage at

a cost per unit time proportional to the underproduction per unit time.

For the sake of simplicity let us assume the cost of moving products *to* storage is the same as the cost of moving products *from* storage, that the maximum output rate is 1, that the minimum output rate is 0, and that $y_0 = 5/6$. We will utilize the same CDF used in Section 8.5.3, and assume that an over or underproduction of 1/8 per unit time results in a cost of \$32 per unit time. This means that $k = 1/2$ (see Figure 8.20). The various functions of the quadratic loss function are calculated in Table 8.19.

Table 8.19: Example 2 Measures

$E[L[\phi(t)]]$	0.0581223
$V[L[\phi(t)]]$	0.00868945
Skewness$[L[\phi(t)]]$	2.08036
Kurtosis$[L[\phi(t)]]$	6.20319

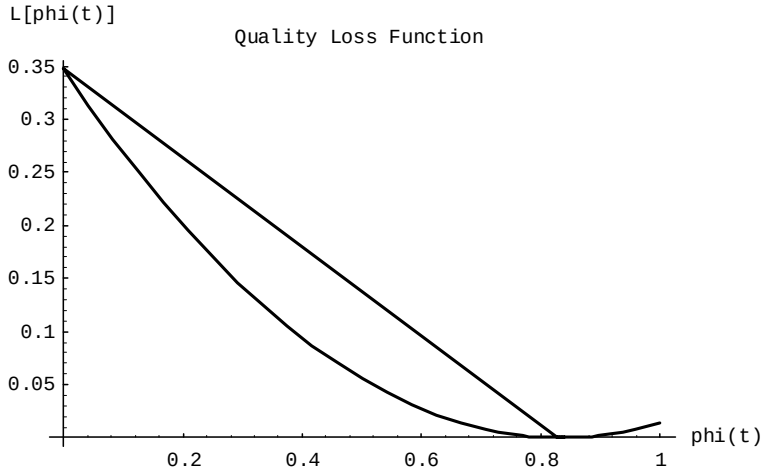


Figure 8.20: Example 2 Quality Loss Function

Chapter 9

FUTURE WORK

One aspect of this field which should be developed more carefully is the process by which particular reliability measures are selected and justified for a given system. In my opinion, there are two potentially fruitful avenues in this area which should be explored:

1. Creating sophisticated "surveys" which determine the relationship between product performance and customer-assessed reliability across similar types of products. The implicit assumption here is that the reliability measure which comes closest to reflecting the experience of the "average customer" *would* be similar across similar products. This would have to be proven.

2. Utilizing advances in cognitive neuroscience to identify potential reliability measures.

Item #2 deserves further explanation. Essentially, I suspect it may be possible to identify likely candidates for reliability measures by examining the sorts of stimuli which humans respond positively and negatively to, based on shared behavioral principles and goals. Dr. Stephen Pinker's work [276] in this area shows considerable promise.

It is true that more efficient algorithms to insure the monotonicity of an interpolation in arbitrary numbers of dimensions may be possible. However, as a practical matter, the activity which might be most valuable to researchers interested in non-binary reliability models is the creation of a national or international repository for

real-world data sets suitable for non-binary reliability analyses. The multistate and continuum reliability literature is conspicuously absent of real-world studies, and the commencement of such studies would almost certainly suggest fruitful areas for further theoretical inquiry.

SOFTWARE FUNCTION INDEX

BIBLIOGRAPHY

[1] J. A. Abraham. An algorithm for the accurate reliability evaluation of triple modular redundancy networks. *IEEE Transactions on Reliability*, 23(7):682–692, 1974.

[2] J. A. Abraham. An improved algorithm for network reliability. *IEEE Transactions on Reliability*, 28(1):58–61, 1979.

[3] K. K. Aggarwal. Comments on: "On the analysis of fault trees". *IEEE Transactions on Reliability*, 25(2):126–127, 1976.

[4] K. K. Aggarwal, K. B. Misra, and J. S. Gupta. A fast algorithm for reliability evaluation. *IEEE Transactions on Reliability*, 24(1):83–85, 1975.

[5] S. H. Ahmad. A simple technique for computing network reliability. *IEEE Transactions on Reliability*, 31(1):41–44, 1982.

[6] R. N. Allan, R. Billinton, and M. F. de Oliveira. An efficient algorithm for deducing the minimal cuts and reliability indices of a general network configuration. *IEEE Transactions on Reliability*, 25(4):226–233, 1976.

[7] R. N. Allan, A. M. Leite da Silva, A. A. Abu-Nasser, and R. C. Burchett. Discrete convolution in power system reliability. *IEEE Transactions on Reliability*, 30(5):452–456, 1981.

[8] R. N. Allan, I. L. Rondiris, and D. M. Fryer. An efficient computational technique for evaluating the cut/tie sets and common-cause failures of complex systems. *IEEE Transactions on Reliability*, 30(2):101–109, 1981.

[9] G. Almassy. Limits of models in reliability engineering. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 364–367, 1979.

[10] S. Arnborg. Reduced state enumeration — another algorithm for reliability evaluation. *IEEE Transactions on Reliability*, 27(2):101–105, 1978.

[11] D. Assaf and B. Levikson. Closure of phase type distributions under operations arising in reliability theory. *Annals of Probability*, 10(1):265–269, 1982.

[12] S. K. Banerjee and S. D. Bhide. A technique to simplify reliability expressions. *IEEE Transactions on Reliability*, 30(3):298–299, 1981.

[13] S. K. Banerjee and K. Rajamani. Parametric representations of probability in two dimensions — a new approach in system reliability evaluation. *IEEE Transactions on Reliability*, pages 56–60, 1972.

[14] S. K. Banerjee and K. Rajamani. Closed form solutions for delta-star and star-delta conversions of reliability networks. *IEEE Transactions on Reliability*, 25(2):118–119, 1976.

[15] R. E. Barlow and S. Iyer. Computational complexity of coherent systems and the reliability polynomial. *Probability in the Engineering and Informational Sciences*, 2:461–469, 1988.

[16] R. E. Barlow and F. Proschan. *Statistical Theory of Reliability and Life Testing: Probability Models*. Holt, Rinehart, and Winston, New York, 1975.

[17] R. E. Barlow and F. Proschan. Some current academic research in system reliability theory. *IEEE Transactions on Reliability*, 25(3):198–202, 1976.

[18] R. G. Bennetts. On the analysis of fault trees. *IEEE Transactions on Reliability*, 24(3):175–185, 1975.

[19] J. E. Biegel. Determination of tie sets and cut sets for a system without feedback. *IEEE Transactions on Reliability*, 26(1):39–42, 1977.

[20] Z. W. Birnbaum and J. D. Esary. Modules of coherent binary systems. *Journal of the Society for Industrial and Applied Mathematics*, 13(2):444–462, 1965.

[21] Z. W. Birnbaum, J. D. Esary, and S. C. Saunders. Multi-component systems and structures and their reliability. *Technometrics*, 3(1):55–77, 1961.

[22] H. W. Block. Dependent components with increasing failure rates and failure rate averages. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 99–101, 1978.

[23] H. W. Block, N. A. Langberg, and T. H. Savits. Repair replacement policies. *Journal of Applied Probability*, 30:194–206, 1993.

[24] H. W. Block, N. A. Langberg, and T. H. Savits. Maintenance comparisons: Block policies. *Journal of Applied Probability*, 27:649–657, 1990.

[25] H. W. Block and T. H. Savits. The IFRA closure problem. *Annals of Probability*, 4(6):1030–1032, 1976.

[26] L. D. Bodin. Optimization procedure for the analysis of coherent structures. *IEEE Transactions on Reliability*, 18(3):118–136, 1969.

[27] L. D. Bodin. Approximations to system reliability using a modular decomposition. *Technometrics*, 12(2):335–344, 1970.

[28] T. B. Boffey and R. J. M. Waters. Calculation of system reliability by algebraic manipulation of probability expressions. *IEEE Transactions on Reliability*, 28(5):358–363, 1979.

[29] G. R. Burdick, J. B. Fussell, D. M. Rasmuson, and J. R. Wilson. Phased mission analysis: A review of new developments and an application. *IEEE Transactions on Reliability*, 26(1):43–49, 1977.

[30] R. W. Butterworth. A set theoretic treatment of coherent systems. *SIAM Journal on Applied Mathematics*, 22(4):590–598, 1972.

[31] J. A. Buzacott. Network approaches to finding the reliability of repairable systems. *IEEE Transactions on Reliability*, 19(4):140–146, 1970.

[32] W.-C. Chan. A generalized reliability function for systems of parallel components. *IEEE Transactions on Reliability*, 17(4):199–201, 1968.

[33] W.-K. Chung. Generalized reliability function for systems of arbitrary configurations. *IEEE Transactions on Reliability*, pages 85–87, 1971.

[34] W. K. Chung and W. C. Chan. A new approach to reliability prediction. *IEEE Transactions on Reliability*, 23(4):252–255, 1974.

[35] C. A. Clarotti. Minimal-cut reliability lower-bound for systems containing standby modules. *IEEE Transactions on Reliability*, 30(3):293–297, 1981.

[36] O. Coudert and J. C. Madre. MetaPrime: An interactive fault-tree analyzer. *IEEE Transactions on Reliability*, 43(1):121–127, 1994.

[37] L. A. Cunningham and N. Singh. The reliability of coherent structures. *IEEE Transactions on Reliability*, pages 276–277, 1973.

[38] J. deMercado, N. Spyratos, and B. A. Brown. A method for calculation of network reliability. *IEEE Transactions on Reliability*, 25(2):71–76, 1976.

[39] R. C. Dubes. Two algorithms for computing reliability. *IEEE Transactions on Reliability*, pages 55–63, 1963.

[40] E. A. Elsayed. *Reliability Engineering*. Addison Wesley, Reading, Massachusetts, 1996.

[41] J. Endrenyi. Algorithms for solving reliability models of systems exposed to a 2-state environment. *IEEE Transactions on Reliability*, 24(4):281–285, 1975.

[42] J. D. Esary and A. W. Marshall. Coherent life functions. *SIAM Journal on Applied Mathematics*, 18(4):810–814, 1970.

[43] J. D. Esary, A. W. Marshall, and F. Proschan. Some reliability applications of the hazard transform. *SIAM Journal on Applied Mathematics*, 18(4):849–680, 1970.

[44] J. D. Esary and F. Proschan. Coherent structures of non-identical components. *Technometrics*, 5(2):191–209, 1963.

[45] J. D. Esary and F. Proschan. Relationship between system failure rate and component failure rates. *Technometrics*, 5(2):183–189, 1963.

[46] R. A. Evans. A generalized limit theorem for reliability. *IEEE Transactions on Reliability*, 18(2):45–46, 1969.

[47] J. P. Gadani and K. B. Misra. Quadrilateral-star transformation: An aid for reliability evaluation of large complex systems. *IEEE Transactions on Reliability*, 31(1):49–50, 1982.

[48] O. Gokcek and G. L. Crellin. Markov analyses of nuclear plant failure dependencies. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 104–109, 1979.

[49] G. E. González-Urdaneta. Comments on: "A reduction technique for obtaining a simplified reliability expression". *IEEE Transactions on Reliability*, 28(1):68–69, 1979.

[50] G. E. González-Urdaneta and B. J. Cory. Variance and approximate confidence limits for probability and frequency of system failure. *IEEE Transactions on Reliability*, 27(4):289–293, 1978.

[51] K. Gopal and S. Rai. Discussion on "A reduction technique for obtaining simplified reliability expression". *IEEE Transactions on Reliability*, 28(1):66, 1979.

[52] A. S. Heger, J. K. Bhat, D. W. Stack, and D. V. Talbott. Calculating exact top-event probabilities using ΣΠ-Patrec. *IEEE Transactions on Reliability*, 44(4):640–644, 1995.

[53] K. D. Heidtmann. Improved method of inclusion-exclusion applied to k-out-of-n systems. *IEEE Transactions on Reliability*, 31(1):36–40, 1982.

[54] W. M. Hirsch, M. Meisner, and C. Boll. Cannibalization in multi-component systems and the theory of reliability. *Naval Research Logistics Quarterly*, 15:331–360, 1968.

[55] J. A. M. Hontelez, H. H. Burger, and D. J. D. Wijnmalen. Optimum condition-based maintenance policies for deteriorating systems with partial information. *Reliability Engineering and System Safety*, 51:267–274, 1996.

[56] A. Høyland and M. Rausand. *System Reliability Theory.* Wiley, New York, 1994.

[57] C. L. Hwang, F. A. Tillman, and W. Kuo. Reliability optimization by generalized Lagrangian-function and reduced-gradient methods. *IEEE Transactions on Reliability*, 28(4):316–319, 1979.

[58] C. L. Hwang, F. A. Tillman, and M. H. Lee. System-reliability techniques for complex/large systems — a review. *IEEE Transactions on Reliability*, 30(5):416–423, 1981.

[59] T. Inagaki, K. Inoue, and H. Akashi. Interactive optimization of system reliability under multiple objectives. *IEEE Transactions on Reliability*, 27(4):264–267, 1978.

[60] R. K. Iyer and T. Downs. A moment approach to evaluation and optimization of complex system reliability. *IEEE Transactions on Reliability*, 27(3):226–229, 1978.

[61] P. A. Jensen and M. Bellmore. An algorithm to determine the reliability of a complex system. *IEEE Transactions on Reliability*, 18(4):169–174, 1969.

[62] L. Kanderhag. Eigenvalue approach for computing the reliability of Markov systems. *IEEE Transactions on Reliability*, 27(5):337–340, 1978.

[63] K. C. Kapur. Optimization in design by reliability. *AIIE Transactions*, 7(2):185–192, 1975.

[64] K. C. Kapur. Reliability bounds in probabilistic design. *IEEE Transactions on Reliability*, 24(3):193–195, 1975.

[65] K. C. Kapur. *Handbook of Industrial Engineering*, chapter 89. Wiley, second edition, 1991.

[66] K. C. Kapur and L. R. Lamberson. *Reliability in Engineering Design*. Wiley, New York, 1977.

[67] P. K. Kapur and K. R. Kapoor. Correction to "A 2-unit warm-standby redundant system with delay and one repair facility". *IEEE Transactions on Reliability*, 27(5):388, 1978.

[68] P. K. Kapur and K. R. Kapoor. Stochastic behavior of some 2-unit redundant systems. *IEEE Transactions on Reliability*, 27(5):382–387, 1978.

[69] Y. Kinberg and E. Shlifer. Some bounds for the reliability of a system with specified failure rate characteristics. *IEEE Transactions on Reliability*, 19(4):191–193, 1970.

[70] E. V. Krishnamurthy and G. Komissar. Computer-aided reliability analysis of complicated networks. *IEEE Transactions on Reliability*, 21(2):86–89, 1972.

[71] A. Kumar, R. M. Pathak, and Y. P. Gupta. Genetic-algorithm-based reliability optimization for computer network expansion. *IEEE Transactions on Reliability*, 44(1):63–72, 1995.

[72] M. O. Locks. The maximum error in system reliability calculations by using a subset of the minimal states. *IEEE Transactions on Reliability*, 20(4):231–234, 1971.

[73] M. O. Locks. Evaluating the KTI Monte Carlo method for system reliability calculations. *IEEE Transactions on Reliability*, 28(5):368–372, 1979.

[74] M. O. Locks. Recursive disjoint products, inclusion-exclusion, and min-cut approximations. *IEEE Transactions on Reliability*, 29(5):368–371, 1980.

[75] M. O. Locks. Fault trees, prime implicants, and noncoherence. *IEEE Transactions on Reliability*, 29(2):130–135, 1980.

[76] K. K. Lee. A compilation technique for exact system reliability. *IEEE Transactions on Reliability*, 30(3):284–288, 1981.

[77] L. L. Levy and A. H. Moore. A Monte Carlo technique for obtaining system reliability confidence limits from component test data. *IEEE Transactions on Reliability*, 16(2):69–72, 1967.

[78] P. M. Lin, B. J. Leon, and T. C. Huang. A new algorithm for symbolic system reliability analysis. *IEEE Transactions on Reliability*, 25(1):2–15, 1976.

[79] D. Magee and A. Refsum. RESIN, a desktop-computer program for finding cut-sets. *IEEE Transactions on Reliability*, 30(5):407–410, 1981.

[80] D. W. McLeavey and J. A. McLeavey. Optimization of system reliability by branch-and-bound. *IEEE Transactions on Reliability*, 25(5):327–329, 1976.

[81] K. B. Misra. A simple approach for constrained redundancy optimization problem. *IEEE Transactions on Reliability*, 21(1):30–34, 1972.

[82] A. H. Moore, H. L. Harter, and R. C. Snead. Comparison of Monte Carlo techniques for obtaining system-reliability confidence limits. *IEEE Transactions on Reliability*, 29(4):327–331, 1980.

[83] D. N. Naik. Estimating the parameters of a 2-out-of-3:F system. *IEEE Transactions on Reliability*, 30(5):464–465, 1981.

[84] T. Nakagawa. On a cumulative damage model with N different components. *IEEE Transactions on Reliability*, 25(2):112–114, 1976.

[85] W. Nelson. Analysis of performance-degradation data from accelerated tests. *IEEE Transactions on Reliability*, 30(2):149–155, 1981.

[86] A. C. Nelson, Jr., J. R. Batts, and R. L. Beadles. A computer program for approximating system reliability. *IEEE Transactions on Reliability*, 19(2):61–65, 1970.

[87] L. Painton and J. Campbell. Genetic algorithms in optimization of system reliability. *IEEE Transactions on Reliability*, 44(2):172–178, 1995.

[88] G. D. M. Pearson. Computer program for approximating the reliability characteristics of acyclic directed graphs. *IEEE Transactions on Reliability*, 26(1):32–38, 1977.

[89] M. J. Phillips. k-out-of-n systems are preferable. *IEEE Transactions on Reliability*, 29(2):166–169, 1980.

[90] A. Prékopa. Sharp bounds on probabilities using linear programming. *Operations Research*, 38(2):227–239, 1990.

[91] S. Rai and K. K. Aggarwal. An efficient method for reliability evaluation of a general network. *IEEE Transactions on Reliability*, 27(3):206–211, 1978.

[92] S. Rai and K. K. Aggarwal. On complementation of pathsets and cutsets. *IEEE Transactions on Reliability*, 29(2):139–140, 1980.

[93] A. Rosenthal. Note on "Closed form solutions for delta-star and star-delta conversion of reliability networks". *IEEE Transactions on Reliability*, 27(2):110–111, 1978.

[94] A. Rosenthal. Approaches to comparing cut-set enumeration algorithms. *IEEE Transactions on Reliability*, 28(1):62–65, 1979.

[95] M. Sakawa. Multiobjective optimization by the surrogate worth trade-off method. *IEEE Transactions on Reliability*, 27(5):311–314, 1978.

[96] A. Satyanarayana and M. K. Chang. Network reliability and the factoring theorem. In *Networks*, volume 13, pages 107–120. Wiley, New York, 1983.

[97] A. Satyanarayana and A. Prabhakar. New topological formula and rapid algorithm for reliability analysis of complex formulas. *IEEE Transactions on Reliability*, 27(2):82–100, 1978.

[98] F. A. Schouten and S. G. Vanneste. Two simple control policies for a multi-component maintenance system. *Operations Research*, 41(6):1125–1136, 1993.

[99] J. Sharma. Algorithm for reliability evaluation of a reducible network. *IEEE Transactions on Reliability*, 25(5):337–339, 1976.

[100] A. W. Shogan. Sequential bounding of the reliability of a stochastic network. *Operations Research*, 34(6):1027–1044, 1976.

[101] A. W. Shogan. A recursive algorithm for bounding network reliability. *IEEE Transactions on Reliability*, 26(5):322–327, 1977.

[102] C. Singh. A cut set method for reliability evaluation of systems having *s*-dependent components. *IEEE Transactions on Reliability*, 29(5):372–375, 1980.

[103] C. Singh and R. Billinton. A new method to determine the failure frequency of a complex system. *IEEE Transactions on Reliability*, 23(4):231–234, 1974.

[104] N. Singh and S. Kumar. Reliability bounds for decomposable multi-component systems. *IEEE Transactions on Reliability*, 29(1):22–23, 1980.

[105] R. Teoste. Digital circuit redundancy. *IEEE Transactions on Reliability*, pages 42–61, 1964.

[106] F. A. Tillman, C.-L. Hwang, and W. Kuo. Optimization techniques for system reliability with redundancy — a review. *IEEE Transactions on Reliability*, 26(3):148–155, 1977.

[107] R. K. Tiwari and M. Verma. An algebraic technique for reliability evaluation. *IEEE Transactions on Reliability*, 29(4):311–313, 1980.

[108] A. Winterbottom and J. C. Verrall. Confidence limits for system reliability: A sequential method. *IEEE Transactions on Reliability*, 20(4):204–211, 1971.

[109] B. P. Zelentsov. Reliability analysis of large nonrepairable systems. *IEEE Transactions on Reliability*, 19(4):152–156, 1970.

[110] R. E. Barlow and F. Proschan. Importance of system components and fault tree analysis. *Stochastic Processes and their Applications*, 3:153–173, 1975.

[111] Z. W. Birnbaum. On the importance of different components in a multicomponent system. In *Multivariate Analysis II*, pages 581–592. Academic Press, New York, 1969.

[112] K. Nakashima and K. Yamato. Variance-importance of system components. *IEEE Transactions on Reliability*, 31(1):99–100, 1982.

[113] B. Natvig. A suggestion of a new measure of importance of system components. *Stochastic Processes and their Applications*, 9:319–330, 1979.

[114] A. M. Abouammah, E. El-Neweihi, and F. Proschan. Schur structure functions. *Probability in the Engineering and Informational Sciences*, 3:581–591, 1989.

[115] R. E. Barlow and A. S. Wu. Coherent systems with multistate components. *Mathematics of Operations Research*, 3(4):275–281, 1978.

[116] W. Borges and F. Rodrigues. An axiomatic characterization of multistate coherent structures. *Mathematics of Operations Research*, 8(3):435–438, 1983.

[117] D. A. Butler. Bounding the reliability of multistate systems. *Operations Research*, 30:530–544, 1982.

[118] G. Cafaro, F. Corsi, and F. Vacca. Multistate Markov models and structural properties of the transition-rate matrix. *IEEE Transactions on Reliability*, 35(2):192–200, 1986.

[119] L. Caldarola. Coherent systems with multistate components. *Nuclear Engineering and Design*, 58:127–139, 1980.

[120] N. Ebrahimi. Multistate reliability models. *Naval Research Logistics Quarterly*, 31:671–680, 1984.

[121] E. El-Neweihi, F. Proschan, and J. Sethuraman. Multistate coherent systems. *Journal of Applied Probability*, 15:675–688, 1978.

[122] E. A. Elsayed and A. Zebib. A repairable multistate device. *IEEE Transactions on Reliability*, 28(1):81–82, 1979.

[123] R. C. Garg and A. Kumar. A complex system with two types of failure and repair. *IEEE Transactions on Reliability*, 26(4):299–300, 1977.

[124] W. S. Griffith. Multistate reliability models. *Journal of Applied Probability*, 17:735–744, 1980.

[125] H. Gupta and J. Sharma. A delta-star transformation approach for reliability evaluation. *IEEE Transactions on Reliability*, 27(3):212–214, 1978.

[126] Y. Hatoyama. Reliability analysis of 3-state systems. *IEEE Transactions on Reliability*, 28(5):386–393, 1979.

[127] A. Karamchandani and C. A. Cornell. Reliability analysis of truss structures with multistate elements II. *Journal of Structural Engineering*, 118(4):910–925, 1992.

[128] J. Karpiński. A multistate system under an inspection and repair policy. *IEEE Transactions on Reliability*, 35(1):76–77, 1986.

[129] E. Korczak. Reliability analysis of multistate monotone systems. In *Proceedings of the European Safety and Reliability Conference*, pages 671–682, Munich, Germany, May 1993. Elsevier Science.

[130] A. Kossow and W. Preuss. Reliability of linear consecutively-connected systems with multistate components. *IEEE Transactions on Reliability*, 44(4):518–522, 1995.

[131] C. T. Lam and R. H. Yeh. Optimal replacement policies for multistate deteriorating systems. *Naval Research Logistics*, 44:303–315, 1994.

[132] A.-A. Mohamed. *Multicriteria Optimization Applied to Multistate Repairable Components*. PhD thesis, University of Oklahoma, 1990.

[133] G. S. Mokaddis and M. L. Tawfek. Some characteristics of a two-dissimilar-unit cold standby redundant system with three modes. *Microelectronics and Reliability*, 36(4):497–503, 1996.

[134] J. D. Murchland. Fundamental concepts and relations for reliability analysis of multi-state systems. In *Proceedings of the 1974 Conference on Reliability and Fault Tree Analysis*, pages 561–618, University of California, Berkeley, September 1975.

[135] B. Natvig. Two suggestions of how to define a multistate coherent system. *Advances in Applied Probability*, 14:434–455, 1982.

[136] F. Ohi and T. Nishida. On multistate coherent systems. *IEEE Transactions on Reliability*, 33(4):284–287, 1984.

[137] A. Pedar and V. V. S. Sarma. Phased-mission analysis for evaluating the effectiveness of aerospace computing-systems. *IEEE Transactions on Reliability*, 30(5):429–437, 1981.

[138] C. L. Proctor, II and C. L. Proctor. Multistate-time dependent system modeling. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 401–403, 1977.

[139] N. Satoh, M. Sasaki, T. Yuge, and S. Yanagi. Reliability of 3-state device systems with simultaneous failures. *IEEE Transactions on Reliability*, 42(3):470–477, 1993.

[140] R. M. Simon. Optimal cannibalization policies for multicomponent systems. *SIAM Journal on Applied Mathematics*, 19(4):700–711, 1970.

[141] M. Veeraraghavan and K. S. Trivedi. A combinatorial algorithm for performance and reliability analysis using multistate models. *IEEE Transactions on Computers*, 43(2):229–234, 1994.

[142] J. Xue and K. Yang. Symmetric relations in multistate systems. *IEEE Transactions on Reliability*, 44(4):689–693, 1995.

[143] J. Yinzhong, X. Fengzhang, G. Jinzhong, and C. Wensu. A study on quantitative analysis of human reliability with inspection. *Reliability Engineering and System Safety*, 44:83–87, 1994.

[144] H. W. Block and T. H. Savits. A decomposition for multistate monotone systems. *Journal of Applied Probability*, 19:391–402, 1982.

[145] R. A. Boedigheimer and K. C. Kapur. Involving the customer in the development and evaluation of multistate reliability models. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 250–255, 1993.

[146] R. A. Boedigheimer and K. C. Kapur. Customer-driven reliability models for multistate coherent systems. *IEEE Transactions on Reliability*, 43(1):46–50, 1994.

[147] M. N. Fardis and C. A. Cornell. Analysis of coherent multistate systems. *IEEE Transactions on Reliability*, 30(2):117–122, 1981.

[148] E. Funnemark and B. Natvig. Bounds for the availabilities in a fixed time interval for multistate monotone systems. *Advances in Applied Probability*, 17:638–665, 1985.

[149] N. L. Hjort, B. Natvig, and E. Funnemark. The association in time of a Markov process with application to multistate reliability theory. *Journal of Applied Probability*, 22:473–479, 1985.

[150] J. C. Hudson. *The Structure and Reliability of Multistate Systems with Multistate Components*. PhD thesis, Wayne State University, Detroit, 1981.

[151] J. C. Hudson and K. C. Kapur. Reliability analysis for multistate systems with multistate components. *IIE Transactions*, 15(2):127–135, 1983.

[152] J. C. Hudson and K. C. Kapur. Modules in coherent multistate systems. *IEEE Transactions on Reliability*, 32(3):183–185, 1983.

[153] J. C. Hudson and K. C. Kapur. Reliability bounds for multistate systems with multistate components. *Operations Research*, pages 153–160, 1985.

[154] F. K. Hwang and Y. C. Yao. Multistate consecutively-connected systems. *IEEE Transactions on Reliability*, 38(4):472–474, 1989.

[155] X. Janan. On multistate system analysis. *IEEE Transactions on Reliability*, 34(4):329–337, 1985.

[156] K. C. Kapur. Reliability engineering and robust design. In *Proceedings of the Ford 2000 Conference on Integration of Quality Methods*, Dearborn, Michigan, November 1994.

[157] J. Malinowski and W. Preuss. Reliability of circular consecutively-connected systems with multistate components. *IEEE Transactions on Reliability*, 44(3):532–534, 1995.

[158] J. Malinowski and W. Preuss. Reliability of reverse-tree-structured systems with multistate components. *Microelectronics and Reliability*, 36(1):1–7, 1995.

[159] J. Malinowski and W. Preuss. Reliability evaluation for tree-structured systems with multistate components. *Microelectronics and Reliability*, 36(1):9–17, 1995.

[160] B. Natvig. Strict and exact bounds for the availabilities in a fixed time interval for multistate monotone systems. *Scandinavian Journal of Statistics*, 20:171–175, 1993.

[161] M. L. Neilsen. Structures for fault-tolerant distributed protocols. In *Proceedings of the Annual IEEE International Conference on Computers and Communications*, pages 84–89, 1994.

[162] B. Natvig and A. Streller. The steady-state behavior of multistate monotone systems. *Journal of Applied Probability*, 21:826–835, 1984.

[163] B. Natvig, S. Sørmo, A. T. Holen, and G. Høgåsen. Multistate reliability theory — a case study. *Advances in Applied Probability*, 18:921–932, 1986.

[164] J. Shao and K. C. Kapur. Multilevel modular decomposition for multistate systems. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 102–107, 1989.

[165] A. P. Wood. Multistate block diagrams and fault trees. *IEEE Transactions on Reliability*, 34(3):236–240, 1985.

[166] M. J. Zuo and M. Liang. Reliability of multistate consecutively-connected systems. *Reliability Engineering and System Safety*, 44:173–176, 1994.

[167] T. Aven. Reliability evaluation of multistate systems with multistate components. *IEEE Transactions on Reliability*, 34(5):473–479, 1985.

[168] T. Aven. Availability evaluation of oil/gas production and transportation systems. *Reliability Engineering*, 18:35–44, 1987.

[169] T. Aven. *Reliability and Risk Analysis*, chapter 4. Elsevier Science, England, 1992.

[170] S. M. Ross. Multivalued state component systems. *Annals of Probability*, 7(2):379–383, 1979.

[171] K. Yu, I. Koren, and Y. Guo. Generalized multistate monotone coherent systems. *IEEE Transactions on Reliability*, 43(2):242–254, 1994.

[172] A. M. Abouammoh and M. A. Al-Kadi. Component relevancy in multistate reliability models. *IEEE Transactions on Reliability*, 40(3):370–379, 1991.

[173] E. El-Neweihi and F. Proschan. Component relevancy in multistate systems. In *Multistate Analysis VI*, pages 203–208. Elsevier Science, 1985.

[174] I. Kuhnert. Comment on: "Component relevancy in multistate reliability models". *IEEE Transactions on Reliability*, 44(1):95–96, 1995.

[175] F. C. Meng. Component-relevancy and characterization results in multistate systems. *IEEE Transactions on Reliability*, 42(3):478–483, 1993.

[176] A. M. Abouammoh and M. A. Al-Kadi. On measures of importance for components in multistate coherent systems. *Microelectronics and Reliability*, 31(1):109–122, 1991.

[177] A. Bossche. Calculation of critical importance for multi-state components. *IEEE Transactions on Reliability*, 36(2):247–249, 1987.

[178] D. A. Butler. A complete importance ranking for components of binary coherent systems, with extensions to multi-state systems. *Naval Research Logistics Quarterly*, 26:565–578, 1979.

[179] M. S. Finkelstein. Once more on measures of importance of system components. *Microelectronics and Reliability*, 34(9):1431–1439, 1994.

[180] P. Kuzminski. Measures of importance in the reliability modeling of multistate component systems. Master's thesis, University of Virginia, 1993.

[181] S.-N. Chiou and V. O. K. Li. Reliability analysis of a communication network with multimode components. *IEEE Journal on Selected Areas in Communications*, 4(7):1156–1161, 1986.

[182] B. S. Dhillon and S. N. Rayapati. A method to evaluate reliability of three-state device networks. *Microelectronics and Reliability*, 26(3):535–554, 1986.

[183] K. Gopal, K. K. Aggarwal, and J. S. Gupta. Reliability analysis of multistate device networks. *IEEE Transactions on Reliability*, 27(3):233–235, 1978.

[184] C.-C. Jane, J.-S. Lin, and J. Yuan. Reliability evaluation of a limited-flow network in terms of minimal cutsets. *IEEE Transactions on Reliability*, 42(3):354–368, 1993.

[185] B. Singh and C. L. Proctor. Reliability analysis of multistate device networks. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 30–35, 1976.

[186] C. R. Tripathy, S. Patra, R. B. Misra, and R. N. Mahapatra. Reliability evaluation of multistage interconnection networks with multi-state elements. *Microelectronics and Reliability*, 36(3):423–428, 1995.

[187] P. K. Varshney, A. R. Joshi, and P.-L. Chang. Reliability modeling and performance evaluation of variable-link capacity networks. *IEEE Transactions on Reliability*, 43(3):378–382, 1994.

[188] T. Aven. On performance measures for multistate monotone systems. *Reliability Engineering and System Safety*, 41:259–266, 1993.

[189] R. Brunelle and K. C. Kapur. Customer-centered reliability methodology. In

Proceedings of the Annual Reliability and Maintainability Symposium, pages 286–292, 1997.

[190] J. A. Jenny. The effect of partial failure modes on reliability analysis. *IEEE Transactions on Reliability*, 18(4):175–180, 1969.

[191] F. A. Tillman, C. H. Lie, and C. L. Hwang. Analysis of pseudo-reliability of a combat tank system and its optimal design. *IEEE Transactions on Reliability*, 25(4):239–242, 1976.

[192] K. S. Trivedi, J. K. Muppala, S. P. Woolet, and B. R. Haverkort. Composite performance and dependability analysis. *Performance Evaluation*, 14:239–242, 1992.

[193] J. Xue and K. Yang. Dynamic reliability analysis of coherent multistate systems. *IEEE Transactions on Reliability*, 44(4):683–688, 1995.

[194] K. Yang and J. Xue. Dynamic reliability measures and life distribution models for multistate systems. *International Journal of Reliability, Quality, and Safety Engineering*, 2(1):79–102, 1995.

[195] L. A. Baxter. Continuum structures I. *Journal of Applied Probability*, 21:802–815, 1984.

[196] L. A. Baxter. Continuum structures II. *Mathematical Proceedings of the Cambridge Philosophical Society*, 99:331–338, 1986.

[197] L. A. Baxter and C. Kim. Bounding the stochastic performance of continuum structure functions I. *Journal of Applied Probability*, 23:660–669, 1986.

[198] H. W. Block and T. H. Savits. Continuous multistate structure functions. *Operations Research*, 32:703–714, 1984.

[199] R. Brunelle and K. C. Kapur. Continuous structure function reliability: An interpolation approach. In *Proceedings of the Annual Industrial Engineering Research Conference*, pages 48–53, 1997.

[200] B. Cappelle and E. E. Kerre. On a possibilistic approach to reliability theory. In *Proceedings of the International Symposium on Uncertainty Modeling and Analysis*, pages 415–418, 1993.

[201] S. N. Iyer and Y. S. Sathe. Redundancy in Barlow-Wu structures. *Journal of the Operational Research Society*, 41(9):843–851, 1990.

[202] C. Kim and L. A. Baxter. Axiomatic characterization of continuum structure functions. *Operations Research Letters*, 6(6):297–300, 1987.

[203] K. Mak. A note on Barlow-Wu structure functions. *Operations Research Letters*, 8:43–44, 1989.

[204] J. Montero. Fuzzy coherent systems. *Kybernetes*, 17(4):28–33, 1988.

[205] J. Montero, J. Tejada, and J. Yáñez. Structural properties of continuum systems. *European Journal of Operational Research*, 45:231–240, 1990.

[206] K. Yang and K. Kapur. Customer-driven reliability: Integration of QFD and robust design. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 339–345, 1997.

[207] K. Yang and J. Xue. Continuous state reliability analysis. In *Proceedings of the Annual Reliability and Maintainability Symposium*, pages 251–257, 1996.

[208] J. Montero. Observable structure functions. *Kybernetes*, 22(2):31–39, 1993.

[209] J. Montero. Reliability bounds for multicriteria systems. *Journal of the Operational Research Society*, 44(10):1025–1034, 1993.

[210] J. Montero, J. Tejada, and J. Yáñez. General structure functions. *Kybernetes*, 23(3):10–19, 1994.

[211] V. Cutello and J. Montero. Reliability structure functions based upon fuzzy numbers. In *Proceedings of the IEEE Conference on Fuzzy Systems*, volume 3, pages 2046–2050, 1994.

[212] O. Kaleva. Fuzzy performance of a coherent system. *Journal of Mathematical Analysis and Applications*, 117:234–246, 1986.

[213] H. W. Block and T. H. Savits. Decompositions for multistate monotone systems. In *Reliability Theory and Models*, pages 231–241. Academic Press, 1984.

[214] R. A. Boedigheimer. *Customer-Driven Reliability Models for Multistate Coherent Systems*. PhD thesis, University of Oklahoma, 1992.

[215] F. C. Meng. Characterizing the Barlow-Wu structure functions. *Naval Research Logistics*, 41:661–668, 1994.

[216] R. Brunelle. *RelPack: A System for Performing Binary, Multistate, and Continuous Reliability Calculations in the Mathematica Environment*, 1996. Published on the Internet.

[217] C.-H. Cheng. Fuzzy consecutive k-out-of-n:F system reliability. *Microelectronics and Reliability*, 34(12):1909–1922, 1994.

[218] G. Hartless and L. Leemis. Computational algebra applications in reliability theory. *IEEE Transactions on Reliability*, 45(3):393–399, 1996.

[219] L. Råde. Reliability survival equivalence. *Microelectronics and Reliability*, 33(6):881–894, 1993.

[220] R. Brunelle. *Mathematics for Engineers and Scientists (5th ed.) by Alan Jeffrey: Mathematica 3.0 Solutions Manual*, 1997. Published on the Internet.

[221] E. Kreyszig and E. J. Norminton. *Advanced Engineering Mathematics: Mathematica Computer Manual*. Wiley, New York, seventh edition, 1995.

[222] S. Wolfram. *The Mathematica Book*. Cambridge University Press, 1996.

[223] Wolfram Research. *Mathematica 3.0 Standard Add-On Packages*. Cambridge University Press, 1996.

[224] P. Alfeld. Scattered data interpolation in three or more variables. In *Mathematical Methods in Computer Aided Geometric Design*, pages 1–33. Academic Press, Boston, 1989.

[225] R. K. Beatson and Z. Ziegler. Monotonicity preserving surface interpolation. *SIAM Journal on Numerical Analysis*, 22(2):401–411, 1985.

[226] R. E. Carlson and T. A. Foley. The parameter R^2 in multiquadric interpolation. *Computers and Mathematics with Applications*, 21:29–42, 1991.

[227] R. E. Carlson and F. N. Fritsch. Monotone piecewise bicubic interpolation. *SIAM Journal on Numerical Analysis*, 22(2):386–400, 1985.

[228] N. Dyn, D. Levin, and S. Rippa. Numerical procedures for surface fitting of scattered data by radial functions. *SIAM Journal on Scientific and Statistical Computing*, 7(2):639–659, 1986.

[229] T. A. Foley and H. Hagen. Advances in scattered data interpolation. *Surveys on Mathematics for Industry*, 4:71–84, 1994.

[230] R. Franke and G. Nielson. Scattered data interpolation and applications: A tutorial and survey. In *Geometric Modeling Methods and Applications*, pages 131–160. Springer-Verlag, New York, 1991.

[231] R. L. Hardy. Multiquadric equations of topography and other irregular surfaces. *Journal of Geophysical Research*, 76(8):1905–1915, 1971.

[232] C. A. Micchelli. Interpolation of scattered data: Distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

[233] G. M. Nielson. Scattered data modeling. *IEEE Computer Graphics and Applications*, pages 60–70, 1993.

[234] G. M. Nielson, T. A. Foley, B. Hamann, and D. Lane. Visualizing and modeling scattered multivariate data. *IEEE Computer Graphics and Applications*, pages 47–54, 1991.

[235] D. Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the ACM National Conference*, pages 517–524, 1968.

[236] A. E. Tarwater. A parameter study of Hardy's multiquadric method for scattered data interpolation. Technical Report UCRL–53670, Lawrence Livermore National Laboratory, 1985.

[237] F. I. Utreras. Constrained surface construction. In *Topics in Multivariate Approximation*, pages 233–254. Academic Press, Boston, 1987.

[238] E. W. Anderson, C. Fornell, and D. R. Lehmann. Customer satisfaction, market share, and profitability: Findings from Sweden. *Journal of Marketing*, 58:53–66, 1994.

[239] M. J. Beckmann and K. C. Kapur. Conjugate duality: Some applications to economic theory. *Journal of Economic Theory*, 5(2):292–302, 1972.

[240] G. Chen and K. C. Kapur. Tolerance design by break-even analysis for reducing variation and cost. *International Journal of Reliability, Quality, and Safety Engineering*, 1(4):445–457, 1994.

[241] D. C. Dorrough. A theoretical analysis of system quality. *IEEE Transactions on Reliability*, 20(3):169–177, 1971.

[242] J. E. Ettlie and M. D. Johnson. Product development benchmarking versus customer focus in applications of quality function deployment. *Marketing Letters*, 5(2):107–116, 1994.

[243] C. Fornell and M. D. Johnson. Differentiation as a basis for explaining customer satisfaction across industries. *Journal of Economic Psychology*, 14:681–696, 1993.

[244] C. Fornell, M. D. Johnson, E. W. Anderson, J. Cha, and B. E. Bryant. The American Customer Satisfaction Index: Nature, purpose, and findings. *Journal of Marketing*, 60:7–18, 1996.

[245] A. Gustafsson and M. D. Johnson. Bridging the quality-satisfaction gap. *Quality Management Journal*, 4(3):27–43, 1997.

[246] International Organization for Standardization. *ISO 8402: Quality Management and Quality Assurance — Vocabulary*, second edition, 1994.

[247] M. D. Johnson, E. W. Anderson, and C. Fornell. Rational and adaptive performance expectations in a customer satisfaction framework. *Journal of Consumer Research*, 21:695–707, 1995.

[248] M. D. Johnson and C. Fornell. A framework for comparing customer satisfaction across individuals and product categories. *Journal of Economic Psychology*, 12:267–286, 1991.

[249] K. C. Kapur. An approach for development of specifications for quality improvement. *Quality Engineering*, 1(1):63–77, 1988.

[250] K. C. Kapur. An integrated customer-focused approach for quality and reliability. In *International Conference on Quality and Reliability*, volume 1, pages 9–17, 1997.

[251] K. C. Kapur and B.-R. Cho. Economic design and development of specifications. *Quality Engineering*, 6(3):401–417, 1994.

[252] R. L. Oliver. A cognitive model of the antecedents and consequences of satisfaction decisions. *Journal of Marketing Research*, 17:460–469, 1980.

[253] M. Phadke. *Quality Engineering Using Robust Design*. Prentice-Hall, 1989.

[254] W. Karwowski and A. Mital. Potential applications of fuzzy sets in industrial safety engineering. *Fuzzy Sets and Systems*, 19:105–120, 1986.

[255] K. S. Park. Fuzzy apportionment of system reliability. *IEEE Transactions on Reliability*, 36(1):129–132, 1987.

[256] P. V. Suresh, D. Chaudhuri, and B. V. A. Rao. Fuzzy-set approach to select maintenance strategies for multistate equipment. *IEEE Transactions on Reliability*, 43(3):451–456, 1994.

[257] G. L. Crellin. The philosophy and mathematics of Bayes' equation. *IEEE Transactions on Reliability*, 21(3):131–135, 1972.

[258] R. G. Easterling. A personal view of the Bayesian controversy in reliability and statistics. *IEEE Transactions on Reliability*, 21(3):186–194, 1972.

[259] D. R. Gimlin and A. M. Breipohl. Bayesian acceptance sampling. *IEEE Transactions on Reliability*, 21(3):176–180, 1972.

[260] W. J. Macfarland. Bayes' equation, reliability, and multiple hypothesis testing. *IEEE Transactions on Reliability*, 21(3):136–147, 1972.

[261] R. L. Racicot. Approximate confidence intervals for reliability of a series system. *IEEE Transactions on Reliability*, 25(4):265–269, 1976.

[262] R. A. Evans. Proper proof of a reliability theorem. *IEEE Transactions on Reliability*, 18(4):205–206, 1969.

[263] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 2. Wiley, New York, second edition, 1971.

[264] S. W. Golomb. Mathematical models: Uses and limitations. *IEEE Transactions on Reliability*, 20(3):130–131, 1971.

[265] P. G. Hoel, S. C. Port, and C. J. Stone. *Introduction to Stochastic Processes*. Houghton Mifflin, Boston, 1972.

[266] A. Jeffrey. *Mathematics for Engineers and Scientists*. Chapman and Hall, London, fifth edition, 1996.

[267] S. Karlin and H. Taylor. *An Introduction to Stochastic Modeling*. Academic Press, San Diego, 1984.

[268] A. Karr. *Probability*. Springer-Verlag, New York, 1993.

[269] M. Loève. *Probability Theory*. D. van Nostrand, Princeton, New Jersey, second edition, 1960.

[270] D. C. Montgomery and G. C. Runger. *Applied Statistics and Probability for Engineers*. Wiley, New York, 1994.

[271] P. Moran. *An Introduction to Probability Theory*. Oxford University Press, 1968.

[272] A. Mosleh and G. Apostolakis. Some properties of distributions useful in the study of rare events. *IEEE Transactions on Reliability*, 31(1):87–94, 1982.

[273] T. Nakagawa and H. Yoda. Relationships among distributions. *IEEE Transactions on Reliability*, 26(5):352–353, 1977.

[274] E. Parzen. *Modern Probability Theory and Its Applications*. Wiley, New York, 1960.

[275] E. Parzen. *Stochastic Processes*. Holden-Day, San Francisco, 1962.

[276] S. Pinker. *How the Mind Works*. W. W. Norton, New York, 1997.

[277] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, second edition, 1992.

[278] W. G. Schneeweiss. Calculating the probability of Boolean expression being 1. *IEEE Transactions on Reliability*, 26(1):16–22, 1977.

[279] C. Singh. Calculating the frequency of Boolean expression being 1. *IEEE Transactions on Reliability*, 26(5):354–358, 1977.

Appendix A

GUIDE TO BIBLIOGRAPHY

A.1 Reference Categories

1. Binary Model
   (a) General References: [1]–[109]
   (b) Component Importance: [110]–[113]
2. Multistate Model
   (a) "Traditional" ($\Omega_i = \{0, 1, \ldots, M\}$): [114]–[143]
   (b) "General" ($\Omega_i = \{0, 1, \ldots, M_i\}$): [144]–[166]
   (c) "Non-Integerial" ($\Omega_i = \{x_{i0}, x_{i1}, \ldots, x_{im_i}\}$): [167]–[170]
   (d) Partially–Ordered: [171]–[171]
   (e) Component Relevancy: [172]–[175]
   (f) Component Importance: [176]–[180]
   (g) Network Reliability: [181]–[187]
3. Multistate and Binary Model Reliability Measures: [188]–[194]
4. Continuum Model: [195]–[207]
5. General Complete Lattice Model: [208]–[210]
6. Fuzzy Logic Model: [211]–[212]

7. Binary, Multistate, and Continuum Models Considered Together: [213]–[215]
8. Mathematica
   (a) Reliability Papers and Packages: [216]–[219]
   (b) Engineering Textbook Solution Sets: [220]–[221]
   (c) Reference Texts: [222]–[223]
9. Scattered Data Interpolation: [224]–[237]
10. Customer Satisfaction, Quality, and Specifications: [238]–[253]
11. Fuzzy Logic in Other Reliability Applications: [254]–[256]
12. Bayesian Approaches to Reliability: [257]–[261]
13. General Probability, Mathematics, and Computer Science: [262]–[279]

A.2 Recommended References

1. Models
   (a) Binary: [21, 66, 56]
   (b) Multistate: [121, 115, 151, 163, 146]
   (c) Continuum: [198, 195, 196, 197, 205, 207]
2. Measures: [192, 145, 188, 194, 189]
3. Mathematical Proofs and Derivations
   (a) Moments from CDF's: [263, page 150]
   (b) Integration of Stochastic Processes: [275, page 79]

Appendix B

COHERENCE DEFINITIONS

B.1 Introduction

This appendix presents the coherence definitions which have appeared most frequently in the reliability literature.

The primary benefit of defining coherence is to facilitate the creation and utilization of theorems involving boundary points.[1] Interestingly enough, it is almost exclusively the "non-decreasing" parts of these coherence definitions which facilitate boundary point analysis rather than their "relevance" requirements; this is ironic because it is only in their definitions of "relevance" that the majority of the multistate model coherence definitions differ from each other.

B.2 Binary Model

Birnbaum et al. [21] coined the phrase "coherent system," and Barlow and Proschan [16] created what is now the standard binary model coherence definition:

1. $\phi(\mathbf{y}) \geq \phi(\mathbf{x})\ \forall \mathbf{y} \geq \mathbf{x}$

2. $\phi(0_i, \mathbf{x}) \neq \phi(1_i, \mathbf{x})$ for some $\mathbf{x} \in S$, $\forall i \in C$

This definition implies that repairing a component which has deteriorated cannot cause the system to deteriorate (i.e. the structure function must be *non-decreasing*),

[1] Though there are certainly theorems unrelated to boundary points which require some sort of coherence, such as the following: For any structure function which is EPS, G_1, or G_2 coherent, $\min \mathbf{x} \leq \phi(\mathbf{x}) \leq \max \mathbf{x}$.

and that each component must in at least one conceivable instance be able to influence the system state (i.e. each component must be *relevant*).

B.3 Multistate Model — Traditional

With minor variations for the sake of consistency, these definitions are given as summarized by Mohamed [132]. The *BW* Class is rephrased as suggested by Borges and Rodrigues [116].

B.3.1 Barlow and Wu (BW Class) [115]

1. For each state vector $\mathbf{x}$ where $\phi(\mathbf{x}) \geq k \geq 1$, there exists a state vector $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ such that $y_i \in \{0, k\}$, $i \in C$, $\mathbf{y} < \mathbf{x}$ and $\phi(\mathbf{y}) \geq k$

2. $\phi(\mathbf{0}) = 0$ and $\phi(\mathbf{M}) = M$

3. For every component $i \in C$ and any state vector $\mathbf{x}$, $\phi(0_i, \mathbf{x}) \leq \phi(M_i, \mathbf{x})$

B.3.2 El-Neweihi, et al. (EPS Class) [121]

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\phi(\mathbf{j}) = j$ for $j \in \Omega$

3. For every level j of every component i, there exists a vector $\mathbf{x}$ such that $\phi(j_i, \mathbf{x}) = j$, while $\phi(k_i, \mathbf{x}) \neq j$ for all $k \neq j$

B.3.3 Butler (B_1 Class) [178]

Originally, this definition was made only for three-state systems: $M_i = M = 2\ \forall i \in C$.

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\phi(\mathbf{0}) = 0$ and $\phi(\mathbf{M}) = M$

3. $\phi(M_i, \mathbf{x}) \neq \phi(0_i, \mathbf{x})$ for $i \in C$ and any vector $\mathbf{x}$

B.3.4 Butler (B_2 Class) [117]

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\phi(\mathbf{0}) = 0$ and $\phi(\mathbf{M}) = M$

3. For every component i, there exists some $\mathbf{x}$ such that $\phi(M_i, \mathbf{x}) > \phi(0_i, \mathbf{x})$

B.3.5 Griffith [124]

According to Mohamed [132], "[Griffith's G_2] definition ... is equivalent to stating that for any component i there exists a vector $\mathbf{x}$ such that $\phi(0_i, \mathbf{x}) < \phi(M_i, \mathbf{x})$." Block and Savits [144] noted that EPS coherence means every state of each component is relevant to the same state of the system, G_1 coherence means every state of each component is relevant to the system, and G_2 coherence means some state of each component is relevant to the system. Requirement #2 of the MMS definition implies $\phi(\mathbf{k}) = k$ for $k \in \Omega$.

Multistate Monotone System (MMS)

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\min \mathbf{x} \leq \phi(\mathbf{x}) \leq \max \mathbf{x}$

G_1 Class

1. $\phi(\mathbf{x})$ is an MMS

2. For any component i and any state j, $j \neq 0$, there exists a vector $\mathbf{x}$ such that $\phi((j-1)_i, \mathbf{x}) < \phi(j_i, \mathbf{x})$

G_2 Class

1. $\phi(\mathbf{x})$ is an MMS

2. For any component i and any state j, there exists a vector $\mathbf{x}$ such that $\phi(j_i, \mathbf{x}) \neq \phi(k_i, \mathbf{x})$ for some state $k \neq j$

B.3.6 Natvig [135]

N_1 Class

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\phi(\mathbf{j}) = j$ for $j \in \Omega$

3. For any component i and every state j, $j \neq 0$, there exists a vector $\mathbf{x}$ such that $\phi(j_i, \mathbf{x}) \geq j$, and $\phi((j-1)_i, \mathbf{x}) \leq j-1$

N_2 Class

A system belongs to the N_2 class if and only if there exist binary coherent structure functions $\phi_j(\mathbf{x})$, $j \neq 0$, such that $\phi(\mathbf{x}) \geq j$ if and only if $\phi_j(I_j(\mathbf{x})) = 1$ for any vector $\mathbf{x}$ and any state $j \neq 0$, where $I_j(\mathbf{x}) = (I_j(x_1), I_j(x_2), \ldots, I_j(x_n))$ and

$$I_j(x_i) = \begin{cases} 1, & x_i \geq j \\ 0, & x_i < j \end{cases}$$

B.3.7 Block and Savits (BS Class) [144]

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\phi(\mathbf{0}) = 0$ and $\phi(\mathbf{M}) = M$

B.3.8 Borges and Rodrigues (BR Class) [116]

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\phi(\mathbf{0}) = 0$ and $\phi(\mathbf{M}) = M$

3. For every i, $i \in C$ there exists a vector $\mathbf{x}$ such that $\phi(0_i, \mathbf{x}) < \phi(M_i, \mathbf{x})$

B.3.9 Ebrahimi [120]

E_1 Class

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\min \mathbf{x} \leq \phi(\mathbf{x}) \leq \max \mathbf{x}$

3. For any component i there exists a state j and a vector $\mathbf{x}$ such that $\phi(j_i, \mathbf{x}) = j$ while $\phi(k_i, \mathbf{x}) \neq j$ for $k \neq j$, $i \in C$, $j \neq 0$

E_2 Class

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\min \mathbf{x} \leq \phi(\mathbf{x}) \leq \max \mathbf{x}$

3. For any component i, there exists a state j and a vector $\mathbf{x}$ such that $\phi(j_i, \mathbf{x}) \geq j$ and $\phi((j-1)_i, \mathbf{x}) \leq j - 1$ for $i \in C$, $j \neq 0$

B.4 Multistate Model — General

B.4.1 Hudson and Kapur (HK Class) [151]

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. For each system state j, there exists at least one vector $\mathbf{x}$ such that $\phi(\mathbf{x}) = j$

3. For all $i \in C$ there exist some $j \in \Omega_i$, $k \in \Omega_i$, and a vector $\mathbf{x}$ such that $\phi(j_i, \mathbf{x}) \neq \phi(k_i, \mathbf{x})$

B.4.2 Ohi and Nishida [136]

ON_1 Class

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. For each component i and all system states s and t, there exist vectors $(j_i, \mathbf{x})$ and $(k_i, \mathbf{x})$ such that $\phi(j_i, \mathbf{x}) = s$ and $\phi(k_i, \mathbf{x}) = t$

ON_2 Class

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. For each component i and all system states s, there exist vectors $(j_i, \mathbf{x})$ and $(k_i, \mathbf{x})$ such that $\phi(j_i, \mathbf{x}) = s - 1$ and $\phi(k_i, \mathbf{x}) = s$

ON_3 Class

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. For each component i and all system states s, there exist vectors $(j_i, \mathbf{x})$ and $(k_i, \mathbf{x})$ such that $\phi(j_i, \mathbf{x}) \neq s$ and $\phi(k_i, \mathbf{x}) = s$

ON_4 Class

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. For each component i and all system states s, there exist vectors $(j_i, \mathbf{x})$ and $(k_i, \mathbf{x})$ such that $\phi(j_i, \mathbf{x}) \leq s - 1$ and $\phi(k_i, \mathbf{x}) \geq s$

ON_5 Class

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. For each component i there exist vectors $(j_i, \mathbf{x})$ and $(k_i, \mathbf{x})$ such that $\phi(j_i, \mathbf{x}) \neq \phi(k_i, \mathbf{x})$

B.4.3 Boedigheimer [214]

If the structure function is specified by lower and upper boundary points, it might be better to define coherence in terms of these points; this is what Boedigheimer's General Multistate Coherent System ($GMCS$) coherence definition accomplishes:

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. S_0 and S_M are not empty.

3. For every component i, there exists a lower boundary point to level k such that $x_i \neq 0$ for some $k \neq 0$ or an upper boundary point to level k such that $x_i \neq M_i$ for some $k \neq M$.

B.5 Multistate Model Coherence Summaries

The multistate model literature has clearly devoted considerable attention to defining various types of coherence: see Table B.1 for a summary of those definitions which have been presented in this appendix.

Some coherence classes are subsets of other classes. The following relationships were given in [132]:

$BW \subseteq EPS \cap N_2 \cap BR$

$EPS \cup N_2 \subseteq N_1 \subseteq G_1 \subseteq G_2 \subseteq BS$

Table B.1: Multistate Model Coherence Definitions

Researchers	Model	Year	Definition
Barlow and Wu	Traditional	1978	BW
El-Neweihi, et al.	Traditional	1978	EPS
Butler	Traditional	1979	B_1
Butler	Traditional	1982	B_2
Griffith	Traditional	1980	G_1
Griffith	Traditional	1980	G_2
Natvig	Traditional	1982	N_1
Natvig	Traditional	1982	N_2
Block and Savits	Traditional	1982	BS
Borges and Rodrigues	Traditional	1983	BR
Ebrahimi	Traditional	1984	E_1
Ebrahimi	Traditional	1984	E_2
Hudson and Kapur	General	1983	HK
Ohi and Nishida	General	1984	ON_1
Ohi and Nishida	General	1984	ON_2
Ohi and Nishida	General	1984	ON_3
Ohi and Nishida	General	1984	ON_4
Ohi and Nishida	General	1984	ON_5
Boedigheimer	General	1992	$GMCS$

These relationships[2] were found by Ebrahimi [120]:

$BW \subseteq EPS \subseteq E_1 \subseteq E_2$

$N_1 \subseteq G_1 \cap E_2 \subseteq G_2$

[2]For the sake of these relations, HK class is considered only for a "traditional" model with the same number of states for each component as for the system.

$E_1 \subseteq HK \subseteq BS$

$G_2 \subseteq HK$

B.6 Continuum Model

B.6.1 Baxter [196]

This definition, which its author refers to as "weak coherence," is similar to Griffith's G_2 Class.

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. $\sup_{\mathbf{x}\in S}[\phi(1_i, \mathbf{x}) - \phi(0_i, \mathbf{x})] > 0$ for each $i \in C$

B.6.2 Boedigheimer [214]

Like his definition for the general multistate case, Boedigheimer's continuum model coherence definition is given in terms of lower and upper boundary points.

1. $\phi(\mathbf{x})$ is non-decreasing in $\mathbf{x}$

2. S_0 and S_1 are not empty

3. For every component i, there exists an $\mathbf{x} \in L_k$ such that $x_i \neq 0$ for some $k \neq 0$ or there exists an $\mathbf{x} \in U_k$ such that $x_i \neq 1$ for some $k \neq 1$

Appendix C

SPECIAL STRUCTURE DEFINITIONS

C.1 Introduction

For every model type, definitions for "parallel," "series," and "k-out-of-n" systems have been proposed in the literature. Philosophically, the essential characteristic of a parallel structure is that the model's best component determines the state of the system, the essential characteristic of a series structure is that the model's nth best (i.e. worst) component determines the state of the system, and the essential characteristic of a k-out-of-n structure is that the model's kth best component determines the state of the system.[1] For all k-out-of-n structure definitions, a series system is an n-out-of-n system and a parallel system is a 1-out-of-n system.

The traditional method of illustrating series and parallel arrangements is shown in Figures C.1 and C.2.

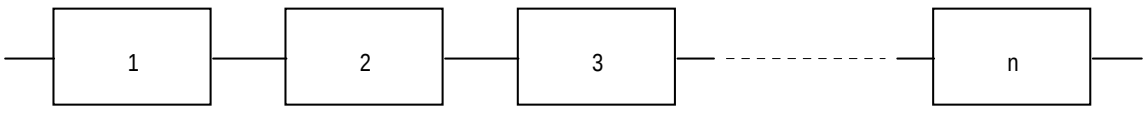


Figure C.1: Series System

[1] Thus, a binary series system is one where the system fails when *any* component fails, and a binary parallel system is one where the system fails only when *all* components fail.

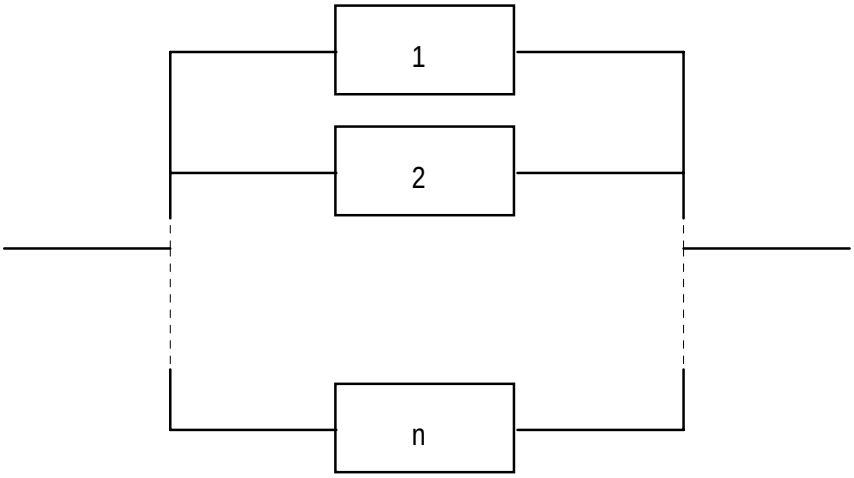


Figure C.2: Parallel System

C.2 Definitions Based on Structures

These are by far the most commonly used definitions. The following are applicable and reasonable for any model where $\Omega = \Omega_i \; \forall i \in C$; Birnbaum et al. [21] first applied these structures to the binary model, and El-Neweihi et al. [121] and Barlow and Wu [115] first applied these structures to the traditional multistate model. Using standard notation, $x_{(i)}$ is the ith order statistic of the set of n components: $x_{(1)} \leq x_{(2)} \leq \ldots \leq x_{(n)}$.

Series[2]: $\phi(\mathbf{x}) = \min \mathbf{x}$

Parallel[3]: $\phi(\mathbf{x}) = \max \mathbf{x}$

k**-out-of-**n: $\phi(\mathbf{x}) = x_{(n-k+1)}$

C.3 Definitions Based on Equivalence Classes

These definitions were created for HK-coherent general multistate models (in which it is possible that $\Omega \neq \Omega_i$) by Hudson and Kapur [151], though they have meaning

[2]For the binary case, $\min \mathbf{x} = \prod_{i=1}^{n} x_i = x_1 x_2 \cdots x_n$.

[3]For the binary case, $\max \mathbf{x} = \coprod_{i=1}^{n} x_i = 1 - (1 - x_1)(1 - x_2) \cdots (1 - x_n)$.

for all model types. In contrast to the definitions in the previous section, there are *many* possible structure functions meeting these definitions which would be considered series, parallel, or k-out-of-n.

Series

1. $S_0 = \{\mathbf{x} \mid \mathbf{x} = (0_i, \mathbf{x})\}$, $\mathbf{x} \in S$, $i \in C$

2. $S_M = \mathbf{M}$

Parallel

1. $S_0 = \mathbf{0}$

2. $S_M = \{\mathbf{x} \mid \mathbf{x} = ((M_i)_i, \mathbf{x})\}$, $\mathbf{x} \in S$, $i \in C$

k-out-of-n

1. $S_0 = \{\mathbf{x} \mid n - k + 1$ or more components are in their minimal states $0\}$, $\mathbf{x} \in S$

2. $S_M = \{\mathbf{x} \mid k$ or more components are in their maximal states $M_i\}$, $\mathbf{x} \in S$

C.4 Definitions Based on Boundary Points

These definitions, introduced by Boedigheimer [214], are also intended for use in situations where it is possible that $\Omega \neq \Omega_i$. They have additional value when the system structure is specified through boundary points, and are valid for all model types:

Series

ϕ has one lower boundary point to level j, $j \neq 0$, and n upper boundary points to level j, $j \neq M$.

Parallel

ϕ has n lower boundary points to level j, $j \neq 0$, and one upper boundary point to level j, $j \neq M$.

k-out-of-n

ϕ has $\binom{n}{k}$ lower boundary points to level j, $j \neq 0$, and $\binom{n}{n-k+1}$ upper boundary points to level j, $j \neq M$.

C.5 Definitions Based on Rounded Structure Function Values

Although the "Equivalence Classes" and "Boundary Points" definitions detailed above do allow for the extension of "parallel" and "series" concepts to the general multistate model, they lose a considerable degree of their original, intuitive character in the process.

A simpler approach would be to retain the definitions for the special structures given in Section C.2, but to require that any given value which they predict be rounded either up, down, or "to the nearest" so that only valid system states may result from any particular combination of the component states. This approach might be especially valuable for non-integerial multistate models with $M = M_i \; \forall i \in C$.

Appendix D

PROBABILITY CALCULATION AND BOUNDS

D.1 Introduction

In this appendix, we review probability calculation and bounding techniques which have been well-documented in the reliability literature. The goal of these techniques is to calculate or approximate probabilities or reliability measures for the system based on the same information about the components.

Many of these bounds are stated in terms of Q_k values. Q_k values can sometimes be used to compare systems directly; one system will be superior to another, all other things being equal, if every Q_k ($k \neq 0$) for that system is greater than for the other. It is worth noting that, for the traditional or general multistate model, one may calculate the expected state of the system based on these values as follows: $E[\phi(\mathbf{X})] = \sum_{j=1}^{M} Q_j$.

D.1.1 Associated Components

Several expressions are given in this appendix for probability bounds on systems whose components are "associated." Associated components are a special category of non-independent components: $X_1, X_2, \ldots, X_n$ are associated if

$$\mathrm{Cov}[g(X_1, X_2, \ldots, X_n), h(X_1, X_2, \ldots, X_n)] \geq 0 \tag{D.1}$$

for all increasing functions g and h. An example of association is a system where several ropes suspend a single object so a failure of one rope increases the strain on the others.

The following statements are true for associated components $X_1, X_2, \ldots, X_n$:

$$P[X_1 > x_1, X_2 > x_2, \ldots, X_n > x_n] \geq \prod_{i=1}^{n} P[X_i > x_i] \tag{D.2}$$

$$P[X_1 \leq x_1, X_2 \leq x_2, \ldots, X_n \leq x_n] \geq \prod_{i=1}^{n} P[X_i \leq x_i] \tag{D.3}$$

For binary associated components these statements have the following equivalent forms:[1]

$$P[\prod_{i=1}^{n} X_i = 1] \geq \prod_{i=1}^{n} P[X_i = 1] \tag{D.4}$$

$$P[\coprod_{i=1}^{n} X_i = 1] \leq \coprod_{i=1}^{n} P[X_i = 1] \tag{D.5}$$

D.2 Exact Calculation

These methods will find precise values for state probabilities of a system based on the state probabilities (or distributions) of its components. Section D.2.1 is applicable to all model types, but Section D.2.2 is applicable only to discrete models.

D.2.1 Direct Enumeration

For discrete models one may find system state probabilities by calculating $P[\phi(\mathbf{X}) = s] = \sum_{\mathbf{x} \in S_s} P[\mathbf{X} = \mathbf{x}]$ (in cases where the components are independent $P[\mathbf{X} = \mathbf{x}] = P[X_1 = x_1]P[X_2 = x_2] \cdots P[X_n = x_n]$, which may simplify computation). For continuum systems, equation (3.3) may be used; when the distributions are specified (or specifiable) by PDF's, $dF_i[x_i, t]$ may be replaced by $f_i[x_i, t]dx_i$, and when the components are not independent $\prod_{i=1}^{n} dF_i[x_i, t]$ should be replaced $dF[x_1, x_2, \ldots, x_n; t]$. The advantage in having independent components for this last calculation is that

[1]Note that for a binary model, $Q_1 = P[\phi(\mathbf{X}) = 1] = E[\phi(\mathbf{X})]$ and $Q_{i1} = P[X_i = 1] = E[X_i]$.

the n-dimensional integral will often break up into the product of n one-dimensional integrals, which can then more easily be solved by numerical integration.

This "brute force" method produces proper results, even for non-coherent systems, but can be computationally intensive; if half of the state vectors in a binary model, for example, lead to the system being in its maximal state, one would have to sum 2^{n-1} probabilities and check all but one of the rest.[2]

D.2.2 Inclusion-Exclusion

These methods produce exact results based on either the lower boundary points or the upper boundary points for a coherent general multistate model [16]. Depending on the number of components and the number of boundary points, this method may or may not be superior to enumeration. The basic theorem used in this method is Feller's Inclusion-Exclusion Principle; it is valid for any v events $E_1, E_2, \ldots, E_v$ and may be stated as follows:

$$P[\bigcup_{j=1}^{v} E_j] = \sum_{j=1}^{v} P[E_j] - \sum_{j<k} P[E_j E_k] + \sum_{j<k<l} P[E_j E_k E_l] - \sum_{j<k<l<m} P[E_j E_k E_l E_m] + \cdots + (-1)^{v+1} P[E_1 E_2 \cdots E_v] \quad \text{(D.6)}$$

For a model with a finite number of lower boundary points to level k, let E_{kj} be the event that $\mathbf{x} \geq L_{kj}$; we may then use (D.6) to compute the right hand side of the following expression:

$$Q_k = P\left[\bigcup_{j=1}^{r_k} E_{kj}\right], \quad k \neq 0 \quad \text{(D.7)}$$

---

[2] Although at least one does not need to separately calculate the boundary points if they were not given.

D.3 Bounds

The bounds in this section all use lower and/or upper boundary points. Because this dissertation's emphasis is on techniques which have practical rather than merely theoretical value, and because of the difficulties with boundary points for continuum models which have been described in previous chapters, these bounds will be presented for discrete cases only (with the exception of (D.9), which requires only one boundary point to the level of interest). Theoretical extensions of these techniques to continuum models are documented in [214].

It should be noted that in the coherent binary case[3], because overall system performance will be worst with a pure series arrangement of the components and best with a pure parallel arrangement of the components, the following bounds are always valid [43]:

$$\prod_{i=1}^{n} Q_{ik} \leq Q_k \leq \coprod_{i=1}^{n} Q_{ik} \tag{D.8}$$

It is assumed throughout this section that, since $Q_0 = 1$, $0 < k \leq M$ for bounds on Q_k.

D.3.1 Trivial Bounds

Trivial bounds [214] are based on a single lower boundary point $\mathbf{y} \in L_k$, and are still valid for associated components:

$$\prod_{i=1}^{n} Q_{i,y_i} \leq Q_k \leq 1 - \prod_{i=1}^{n} (1 - Q_{i,y_i}) \tag{D.9}$$

D.3.2 Path/Cut Bounds

Path/cut bounds [144] find the upper bound from the lower boundary points and the lower bound from the upper boundary points. They are not in every case narrower than trivial bounds. Notation used below is for the general multistate model.

---

[3] and in some non-binary cases, depending on what coherence definition is followed

Independent Components:

$$\prod_{\mathbf{x}\in U_{k-1}} \coprod_{(i,j)\in U_{k-1}(\mathbf{x})} Q_{i,j+1} \leq Q_k \leq \coprod_{\mathbf{x}\in L_k} \prod_{(i,j)\in L_k(\mathbf{x})} Q_{i,j} \tag{D.10}$$

Associated Components:

$$\prod_{\mathbf{x}\in U_{k-1}} P\left[\bigcup_{(i,j)\in U_{k-1}(\mathbf{x})} \{X_i > j\}\right] \leq Q_k \leq \coprod_{\mathbf{x}\in L_k} P\left[\bigcap_{(i,j)\in L_k(\mathbf{x})} \{X_i > j-1\}\right] \tag{D.11}$$

D.3.3 Min/Max Bounds

Min/max bounds [144] find the lower bound from the lower boundary points and the upper bound from the upper boundary points. They are in every case narrower than (or at least equal to, in degenerate cases) trivial bounds, but are not necessarily narrower than path/cut bounds. Notation used below is for the general multistate model.

Non-Associated Components:

$$\max_{\mathbf{x}\in L_k}\{\prod_{(i,j)\in L_k(\mathbf{x})} Q_{i,j}\} \leq Q_k \leq \min_{\mathbf{x}\in U_{k-1}}\{\coprod_{(i,j)\in U_{k-1}(\mathbf{x})} Q_{i,j+1}\} \tag{D.12}$$

Associated Components:

$$\max_{\mathbf{x}\in L_k} P[\bigcap_{(i,j)\in L_k(\mathbf{x})} \{X_i > j-1\}] \leq Q_k \leq \min_{\mathbf{x}\in U_{k-1}} P[\bigcup_{(i,j)\in U_{k-1}(\mathbf{x})} \{X_i > j\}] \tag{D.13}$$

D.3.4 Combination Bounds

Since, according to Boedigheimer [214], "Upper boundary points generally provide a tighter bound for mutually independent components with large probabilities in the

higher states, and the lower boundary points generally provide a tighter bound for mutually independent components with large probabilities in the lower states," it is logical that one could combine the bounds produced by the path/cut and min/max methods to obtain better results. Of the bounds calculated by path/cut and min/max methods, one would choose the larger of the two lower bounds and the smaller of the two upper bounds to form the combined bounds.

D.3.5 Inclusion/Exclusion Bounds

For inclusion/exclusion bounds [135], one chooses a calculation depth based on competing needs for speed and accuracy. These bounds are not constrained to lie between 0 and 1, and furthermore do not always steadily improve as one adds more terms. The second order, fourth order, jth order (j even), and exact (r_kth order) terms are given below; each Σ_i is the ith term on the right-hand side of (D.6).

$$\begin{aligned}
\text{2nd Order:} \quad & \Sigma_1 - \Sigma_2 \le Q_k \le \Sigma_1 \\
\text{4th Order:} \quad & \Sigma_1 - \Sigma_2 + \Sigma_3 - \Sigma_4 \le Q_k \le \Sigma_1 - \Sigma_2 + \Sigma_3 \\
j\text{th Order:} \quad & \sum_{i=1}^{j} (-1)^{i+1} \Sigma_i \le Q_k \le \sum_{i=1}^{j-1} (-1)^{i+1} \Sigma_i \\
\text{Exact:} \quad & Q_k = \Sigma_1 - \Sigma_2 + \Sigma_3 - \cdots + (-1)^{r_k+1} \Sigma_{r_k}
\end{aligned}$$

Appendix E

MISCELLANEOUS THEOREMS AND DEFINITIONS

E.1 Component Redundancy

The following were proved for binary coherent systems in [43], and are also valid for any other model type with a non-decreasing structure function. The following new notation is used in this section:

$$a \vee b \equiv \max\{a, b\}$$
$$\mathbf{x} \vee \mathbf{y} \equiv (x_1 \vee y_1, x_2 \vee y_2, \ldots, x_n \vee y_n)$$
$$a \wedge b \equiv \min\{a, b\}$$
$$\mathbf{x} \wedge \mathbf{y} \equiv (x_1 \wedge y_1, x_2 \wedge y_2, \ldots, x_n \wedge y_n)$$

E.1.1 Parallel

$$\phi(\mathbf{x} \vee \mathbf{y}) \geq \phi(\mathbf{x}) \vee \phi(\mathbf{y}) \tag{E.1}$$

In other words, if two copies of each component are available, it is better to build one structure where each component is in parallel with its twin rather than to build two identical structures which are themselves in parallel. This implies that redundancy is more effective at the component level than at the system level. Equality holds in the above expression when the structure function $\phi(\mathbf{x})$ was parallel to begin with.

E.1.2 Series

$$\phi(\mathbf{x} \wedge \mathbf{y}) \leq \phi(\mathbf{x}) \wedge \phi(\mathbf{y}) \tag{E.2}$$

In other words, if two copies of each component are available, it is better to build two identical structures which are themselves in series rather than to build one structure where each component is in series with its twin. Equality holds in the above expression when the structure function $\phi(\mathbf{x})$ was series to begin with.

E.2 Component Importance

Generally, it is considered prudent to concentrate real-life improvement efforts on the most important components of a system. Measures such as these, if calculable, may help guide engineering decisions.

E.2.1 Structural

Intuitively, one component might be considered more important than another if there is a greater proportion of state vectors in which it may affect the state of the system by being repaired or decaying. A measure for this "structural importance" can be calculated based solely on knowledge of the system's structure function, without any knowledge of the components' stochastic properties.

Boedigheimer [214] gave a general multistate equivalent of this originally binary expression as follows:

$$I_\phi(i) = \frac{1}{\prod_{j\neq i}(M_j+1)} \sum_{\{\mathbf{x}|\mathbf{x}_i=M_i\}} N(\mathbf{x}) \tag{E.3}$$

where

$$N(\mathbf{x}) = \begin{cases} 1, & \phi((M_i)_i, \mathbf{x}) - \phi(0_i, \mathbf{x}) > 0 \\ 0, & \text{otherwise} \end{cases} \tag{E.4}$$

Boedigheimer also extended (E.3) to the continuum case:[1]

$$I_\phi(i) = \int_{\mathbf{x}|x_i=1} N(\mathbf{x})\, d\mathbf{x} \tag{E.5}$$

[1] Please note that a component can actually be relevant for a continuum model, but irrelevant for a binary model with the same structure function.

E.2.2 Performance

Performance Importance (referred to as "Reliability Importance" in the binary model literature) is analogous to Structural Importance, but takes into account the components' stochastic properties. The Performance Importance of general multistate component i at state j, as defined by El-Neweihi et al. [121], is as follows:

$$I_R(i,j) = E[\phi(j_i, \mathbf{X})] - E[\phi(0_i, \mathbf{X})],\ j \neq 0 \tag{E.6}$$

For a parallel binary system, the component with the highest reliability will be most important; for a series binary system, the component with the lowest reliability will be the most important.

Appendix F

DYNAMIC PROPERTIES

This appendix summarizes basic results on the dynamic properties of reliability models. Dynamic properties concern the relationship between the lifetime distributions for components and the lifetime distribution for the system. It will be assumed throughout this appendix that the components are mutually independent and non-repairable, that systems and components are in their best states at $t = 0$, and that the system's structure function is coherent.

F.1 Binary Model

F.1.1 Notation

Let T be the random variable for the lifetime of the system[1] (the time at which the system state reverts to 0), and let $f(t)$ be the PDF for T. The following standard notation may be used (subscripts may be added when these quantities are computed

[1]Note that $f(t)$ and $F(t)$, in this section, are the PDF and CDF for the *time to failure* (in a binary sense) rather than for the system state.

for components):

$$F(t) \equiv P[T \leq t] = \int_0^t f(\tau)\, d\tau \tag{F.1}$$

$$R(t) \equiv P[T > t] = 1 - F(t) \tag{F.2}$$

$$h(t) \equiv \frac{f(t)}{R(t)} \tag{F.3}$$

$$H(t) \equiv \int_0^t h(\tau)\, d\tau \tag{F.4}$$

$$\mu \equiv \int_0^\infty t f(t)\, dt \tag{F.5}$$

F.1.2 Lifetime Distribution Classes

Please see Table F.1 for a summary of lifetime distribution class definitions which have appeared in the binary model literature.

F.1.3 Lifetime Distribution Closure

Barlow and Proschan [16] examined the issue of lifetime distribution closure — when operations on components of a certain class always result in a lifetime distribution of that same class; the operations they examined were forming coherent systems, summing lifetime distributions (convolutions), and forming linear combinations of lifetime distributions. Their results are summarized in Table F.2; we can conclude from this table, for example, that a coherent system whose components are all IFRA will be IFRA, but a coherent system whose components are all IFR may or may not be IFR.

F.2 Bounding System Lifetime Distributions

If one can assume all the unknown component lifetime distributions are IFR with known means μ_i (where $i \in C$), one can obtain useful bounds on system probabilities by realizing the exponential distribution is the limiting distribution for the

Table F.1: Binary Model Lifetime Distribution Classes

Class	Full Name	Characteristic
IFR	Increasing Failure Rate	$h(t)$ is a non-decreasing function
IFRA	Increasing Failure Rate on the Average	$H(t)/t$ is a non-decreasing function
NBU	New Better than Used	$R(t+x) \leq R(t)R(x)$ for $t \geq 0$, $x \geq 0$
NBUE	New Better than Used in Expectation	$\int_t^\infty R(\tau)\,d\tau \leq \mu R(t)$
DFR	Decreasing Failure Rate	$h(t)$ is a nonincreasing function
DFRA	Decreasing Failure Rate on the Average	$H(t)/t$ is a nonincreasing function
NWU	New Worse than Used	$R(t+x) \geq R(t)R(x)$ for $t \geq 0$, $x \geq 0$
NWUE	New Worse than Used in Expectation	$\int_t^\infty R(\tau)\,d\tau \geq \mu R(t)$

IFR class [16]; a lower bound on $R(t)$, valid for $t < \min\{\mu_1, \mu_2, \ldots, \mu_n\}$, can then be shown to be $\phi(e^{-t/\mu_1}, e^{-t/\mu_2}, \ldots, e^{-t/\mu_n})$. A similar result is available for the multistate model [214].

F.3 Multistate Model

F.3.1 Lifetime Distribution Classes and Closure

IFRA

Ross [170] defined IFRA such that the length of time for the component or system to reach or go below *each* state j is IFRA (as defined for the binary model). He proved

Table F.2: Binary Model Lifetime Distribution Closure

Lifetime Dist.	Coherent Systems	Convolutions	Linear Combs.
IFR	Open	Closed	Open
IFRA	Closed	Closed	Open
NBU	Closed	Closed	Open
NBUE	Open	Closed	Open
DFR	Open	Open	Closed
DFRA	Open	Open	Closed
NWU	Open	Open	Open
NWUE	Open	Open	Undetermined

closure with regard to the formation of coherent systems for IFRA components under this expanded definition. An equivalent result was proved for the general multistate model by Hudson [150].

NBU

El-Neweihi et al. [121] defined NBU such that the length of time for the component or system to reach or go below *each* state j is NBU (as defined for the binary model). He proved closure with regard to the formation of coherent systems for NBU components under this expanded definition. An equivalent result was proved for the general multistate model by Hudson [150].

F.4 Continuum Model

Baxter [195] proved (using definitions of IFRA and NBU identical to those given in Section F.3.1) closure with regard to the formation of coherent systems for these continuum IFRA and NBU classes.

Appendix G

LAPLACE TRANSFORM REFERENCE

Definition:

$$\mathcal{L}[f(t)] = f^*(z) = \int_0^\infty e^{-zt} f(t)\, dt, \qquad f(t) \text{ defined on } [0, \infty)$$

Properties:

$$\mathcal{L}[\alpha f_1(t) + \beta f_2(t)] = \alpha \mathcal{L}[f_1(t)] + \beta \mathcal{L}[f_2(t)] \tag{G.1}$$

$$\mathcal{L}[f(t-\alpha)] = e^{-\alpha z} \mathcal{L}[f(t)] \tag{G.2}$$

$$\mathcal{L}[e^{\alpha t} f(t)] = f^*(z-\alpha) \tag{G.3}$$

$$\mathcal{L}[f'(t)] = z\mathcal{L}[f(t)] - f(0) \tag{G.4}$$

$$\mathcal{L}[\int_0^t f(u)\, du] = \mathcal{L}[f(t)]/z \tag{G.5}$$

$$\mathcal{L}[\int_0^t f_1(t-u) f_2(u)\, du] = \mathcal{L}[f_1(t)] \cdot \mathcal{L}[f_2(t)] \tag{G.6}$$

$$\lim_{z\to\infty} z f^*(z) = \lim_{t\to 0} f(t) \tag{G.7}$$

$$\lim_{z\to 0} z f^*(z) = \lim_{t\to\infty} f(t) \tag{G.8}$$

see Table G.1 for several common Laplace transforms.

Table G.1: Table of Laplace Transforms

$f(t)$	$\mathcal{L}[f(t)]$	$f(t)$	$\mathcal{L}[f(t)]$
1	$1/z$	t^α	$\Gamma(\alpha+1)/z^{\alpha+1}$
t	$1/z^2$	$e^{\alpha t}$	$1/(z-\alpha)$
t^2	$2!/z^3$	$e^{\alpha t}t^n$	$n!/(z-\alpha)^{n+1}$
t^n	$n!/z^{n+1}$		

Appendix H

CONTINUOUS DISTRIBUTION REFERENCE

The following represents the definitions and notation for various PDF's, as used both by *Mathematica* and by this dissertation. Note that $\mathrm{B}(a,b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$.

DISTRIBUTION	$f(t)$
Beta$[p,q]$	$\frac{(1-t)^{q-1}t^{p-1}}{\mathrm{B}(p,q)}$
Cauchy$[a,b]$	$\frac{1}{b\pi((t-a)^2/b^2+1)}$
Chi$[n]$	$\frac{2^{1-n/2}e^{-t^2/2}t^{n-1}}{\Gamma(n/2)}$
ChiSquare$[n]$	$\frac{2^{-n/2}e^{-t/2}t^{n/2-1}}{\Gamma(n/2)}$
Exponential$[\lambda]$	$e^{-t\lambda}\lambda$
ExtremeValue$[\alpha,\beta]$	$\frac{\exp(((\alpha-t)/\beta)^{-\exp((\alpha-t)/\beta)})}{\beta}$
FRatio$[n_1,n_2]$	$\frac{n_1^{n_1/2}n_2^{n_2/2}t^{n_1/2-1}(n_2+n_1t)^{-(n_1+n_2)/2}}{\mathrm{B}(n_1/2,n_2/2)}$
Gamma$[\alpha,\beta]$	$\frac{e^{-t/\beta}t^{\alpha-1}\beta^{-\alpha}}{\Gamma(\alpha)}$
HalfNormal$[\theta]$	$\frac{2e^{-(t^2\theta^2)/\pi}\theta}{\pi}$
Laplace$[\mu,\beta]$	$e^{-\frac{\lvert t-\mu\rvert}{2\beta^2}}$
Logistic$[\mu,\beta]$	$\frac{e^{-\frac{t-\mu}{\beta}}}{\left(1+e^{-\frac{t-\mu}{\beta}}\right)^2\beta}$
LogNormal$[\mu,\sigma]$	$\frac{e^{-(\ln(t)-\mu)^2/(2\sigma^2)}}{\sqrt{2\pi}t\sigma}$

$$\begin{aligned}
\text{Normal}[\mu,\sigma] &\qquad \frac{e^{-(t-\mu)^2/(2\sigma^2)}}{\sqrt{2\pi}\sigma} \\
\text{Rayleigh}[\sigma] &\qquad \frac{e^{-t^2/(2\sigma^2)}t}{\sigma^2} \\
\text{StudentT}[n] &\qquad \frac{\left(\frac{n}{t^2+n}\right)^{(n+1)/2}}{\sqrt{n}\,\mathrm{B}(n/2,1/2)} \\
\text{Uniform}[\min,\max] &\qquad \frac{1}{\max-\min} \\
\text{Weibull}[\alpha,\beta] &\qquad e^{-(t/\beta)^\alpha}t^{\alpha-1}\alpha\beta^{-\alpha}
\end{aligned}$$

When necessary, we may take advantage of the following relations:

$$\begin{aligned}
\text{Gamma}[n/2,2] &= \text{ChiSquare}[n] \\
\text{Gamma}[1,1/\lambda] &= \text{Exponential}[\lambda] \\
\text{Weibull}[1,1/\lambda] &= \text{Exponential}[\lambda] \\
\text{Weibull}[1,\beta] &= \text{Gamma}[1,\beta] \\
\text{Weibull}[2,\sqrt{2}\sigma] &= \text{Rayleigh}[\sigma]
\end{aligned}$$

Let us assume that the customer specifies the mode $m > 0$ of a $U(t)$ lifetime weighting function. The customer's options in terms of the PDF's given at the be-

ginning of this appendix are:

$$\text{Chi}[m^2+1]$$

$$\text{ChiSquare}[m+2]$$

$$\text{FRatio}\left[n_1, \left(\frac{1-2/n_1}{2m}-\frac{1}{2}\right)^{-1}\right], \quad n_1>2, m<1$$

$$\text{Gamma}\left[\alpha, \frac{m}{\alpha-1}\right], \quad \alpha>1$$

$$\text{LogNormal}\left[\mu, \sqrt{\mu-\ln m}\right]$$

$$\text{Rayleigh}[m]$$

$$\text{Uniform}[\min \le m, \max \ge m]$$

$$\text{Weibull}\left[\alpha, \frac{m}{((\alpha-1)/\alpha)^{1/\alpha}}\right], \quad \alpha>1$$

Rayleigh and LogNormal can never have a mode of 0, while Exponential and HalfNormal must always have a mode of 0. Other PDF's which can be valuable for $m=0$ cases are FRatio with $n_1 \le 2$, ChiSquare with $n_1 \le 2$, Weibull with $\alpha \le 1$, Gamma with $\alpha \le 1$, Chi with $n \le 1$, and Uniform with $\min = 0$.

Two different types of truncation are used in this dissertation for continuous distributions in continuum model examples. A "hard-truncated" continuous distribution, abbreviated in the text as "h-truncated" or "ht-[DistributionName]," is the following function of the original CDF F[x]:

$$F_{ht}[x] = \begin{cases} 1, & x \ge 1 \\ F[x], & 0 \le x < 1 \\ 0, & x < 0 \end{cases} \tag{H.1}$$

A "soft-truncated" continuous distribution, abbreviated in the text as "s-truncated"

or "*st*-[DistributionName]," is the following function of the original CDF F[x]:

$$F_{st}[x] = \begin{cases} 1, & x > 1 \\ \frac{F[x]-F[0]}{F[1]-F[0]}, & 0 \leq x \leq 1 \\ 0, & x < 0 \end{cases} \tag{H.2}$$

A "soft-truncated" PDF may also be defined:

$$f_{st}[x] = \begin{cases} 0, & x > 1 \\ \frac{f[x]}{F[1]-F[0]}, & 0 \leq x \leq 1 \\ 0, & x < 0 \end{cases} \tag{H.3}$$

Appendix I

SOFTWARE DOCUMENTATION AND TUTORIALS

Original *Mathematica* [222] software packages which can perform all the calculations described in this dissertation are available on the accompanying disk and printed in Appendix J. This software is unique in that it both allows for non-binary reliability analyses and can often return answers in symbolic as well as numerical form. This appendix gives the user a basic understanding of these new packages' capabilities and illustrates how they were used to generate the graphs, calculations, etc. which appear earlier in this dissertation.

There are definite precedents for using *Mathematica* to perform advanced reliability calculations; some of the best efforts along these lines are documented in [218, 217, 219].

In the segment of this appendix titled "Mixed Model Tutorial," the "attached sheet" is in fact page 115 of this dissertation, which contains Figure 8.15.

Introduction to RelPack 2.0

■ Purpose and Background

The purpose of this section is to introduce the user to the main features and syntax for RelPack 2.0. RelPack is collection of *Mathematica* packages, which when taken together provide a variety of functions for performing reliability model-building and analysis with binary, multistate, and continuous reliability models.

It is necessary to have a certain level of knowledge concerning *Mathematica* to make effective use of RelPack. RelPack's goal is to provide a collection of functions that can be used, combined, and extended for the purposes of reliability analyses. However, the underlying syntax used for all these functions is that of the *Mathematica* language, and without some knowledge of it this environment cannot be utilized to its fullest extent. For more information on *Mathematica*, the user is referred to the standard defining text of the language [*The Mathematica Book* by Stephen Wolfram (1996)] and the *Mathematica* web site [www.wolfram.com].

RelPack was written as an essential element of the author's doctoral dissertation work.

■ Package Basics

■ Loading the RelPack Environment

After the *Mathematica* interpreter has been invoked, all the functionality of RelPack may be loaded by using the following command:

```
Needs["LoadReliability'"]
```

Alternatively, one may load separate groups of functions (called "Packages") individually instead of loading the entire set of them. Loading packages individually may be necessary in cases where memory is scarce.

The DeterministicAnalysis Package

■ General Comments and Definitions

In general, the functions in this package are designed to assist with multistate, rather than continuum systems. The vector comparison operators LessOrEqualQ, LessQ, GreaterOrEqualQ, and GreaterQ are the only functions which will have value for non-discrete models. The emphasis in this package is on the manipulation of minimal paths and cuts, and on coherence verification.

```
phi$Max[x_] := Max[x]
```

■ Function Documentation

■ LessOrEqualQ[x,y]

This function will return "True" if the vector x is less than or equal to the vector y, "False" otherwise.

```
LessOrEqualQ[{0,0,1},{1,1,2}]
```

```
True
```

■ LessQ[x,y]

This function will return "True" if the vector x is less than the vector y, "False" otherwise.

```
LessQ[{1,1,1},{1,1,1}]
```

```
False
```

■ GreaterOrEqualQ[x,y]

This function will return "True" if the vector x is greater than or equal to the vector y, "False" otherwise.

```
GreaterOrEqualQ[{1,1,1},{1,1,1}]
```

```
True
```

■ GreaterQ[x,y]

This function will return "True" if the vector x is greater than the vector y, "False" otherwise.

```
GreaterQ[{1,1,2},{1,1,1}]
```

```
True
```

■ VectorSpace[p]

This function will return the list of all possible component state vectors, given the list of possible states for each vector, p.

```
VectorSpace[{{0,1,2},{0,1,2,3},{0,1}}]
```

```
{{0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1}, {0, 2, 0}, {0, 2, 1},
 {0, 3, 0}, {0, 3, 1}, {1, 0, 0}, {1, 0, 1}, {1, 1, 0}, {1, 1, 1},
 {1, 2, 0}, {1, 2, 1}, {1, 3, 0}, {1, 3, 1}, {2, 0, 0}, {2, 0, 1},
 {2, 1, 0}, {2, 1, 1}, {2, 2, 0}, {2, 2, 1}, {2, 3, 0}, {2, 3, 1}}
```

■ NonDecreasingQ[p,phi]

This function will return "True" if the given structure function is non-decreasingon the space of possible component state vectors, "False" otherwise.

```
NonDecreasingQ[{{0,1,2},{0,1,2,3},{0,1}},phi$Max]
```

```
True
```

■ ProperLimitsQ[p,phi,fphi]

This function will return "True" if the given structure function maps the maximal and minimal component state vectors into the maximal and minimal system states, respectively. "False" is returned otherwise.

```
ProperLimitsQ[{{0,1,2},{0,1,2,3},{0,1}},
              phi$Max,{0,1,2,3}]
```

```
True
```

■ ReleventComponentsQ[p,phi]

This function will return "True" if every component is relevent, "False" otherwise. A component is deemed relevent if there is some component state vector for which some change in this component will cause a change in the system state.

```
ReleventComponentsQ[{{0,1,2},{0,1,2,3},{0,1}},phi$Max]
```

```
True
```

■ CoherentQ[p,phi,fphi]

This function will return "True" if the given multistate system is coherent, false otherwise. A system is deemed coherent if it passes the ReleventComponentsQ,ProperLimitsQ,and NonDecreasingQtests.

```
CoherentQ[{{0,1,2},{0,1,2,3},{0,1}},phi$Max,{0,1,2,3}]
```

```
True
```

■ LBPFromStructure[p,phi,fphi]

This function will return the lower boundary points to each level, under the assumption that the system is coherent. For each boundary point, the first part of the list gives the component vector for that boundary point, the second gives the system state that vector is a boundary point for, the third defines whether that is an upper or lower boundary point, and the last will indicate whether that boundary point is "Real" or "Virtual". A boundary point is termed virtual if the system state associated with that component state vector is not equal to the system state that component is a boundary point for. Please note that the Virtual boundary points are necessary to retain, if one wishes to reconstruct the original structure function accurately based only on the boundary points. Compare the two examples below for clarification of this point. The situation with UBPFromStructureis exactly analagous.

```
lbps=LBPFromStructure[{{0,1,2},{0,1,2,3},{0,1}},
   phi$Max,{0,1,2,3}]
```

```
{{{0, 0, 1}, 1, Lower, Real}, {{0, 1, 0}, 1, Lower, Real},
 {{1, 0, 0}, 1, Lower, Real}, {{0, 2, 0}, 2, Lower, Real},
 {{2, 0, 0}, 2, Lower, Real}, {{0, 3, 0}, 3, Lower, Real}}
```

```
LBPFromStructure[{{0,2},{0,2,3},{0,2}},
   phi$Max,{0,1,2,3}]
```

```
{{{0, 0, 2}, 1, Lower, Virtual}, {{0, 2, 0}, 1, Lower, Virtual},
 {{2, 0, 0}, 1, Lower, Virtual}, {{0, 0, 2}, 2, Lower, Real},
 {{0, 2, 0}, 2, Lower, Real}, {{2, 0, 0}, 2, Lower, Real},
 {{0, 3, 0}, 3, Lower, Real}}
```

■ UBPFromStructure[p,phi,fphi]

This function will return the upper boundary points to each level, under the assumption that the system is coherent. See LBPFromStructurefor further discussion.

```
ubps=UBPFromStructure[{{0,1,2},{0,1,2,3},{0,1}},
   phi$Max,{0,1,2,3}]
```

```
{{{0, 0, 0}, 0, Upper, Real}, {{1, 1, 1}, 1, Upper, Real},
 {{2, 2, 1}, 2, Upper, Real}}
```

```
UBPFromStructure[{{0,1,3},{0,1,3},{0,3}},
   phi$Max,{0,1,2,3}]
```

```
{{{0, 0, 0}, 0, Upper, Real}, {{1, 1, 0}, 1, Upper, Real},
 {{1, 1, 0}, 2, Upper, Virtual}}
```

■ CUVUpperBound[n,m]

This function will return the Xue and Yang [1995] upper bound on the number of Critical Upper Vectors for each level j. It is assumed that the system and each and each of the n components have the same number of possible states: m+1.

```
CUVUpperBound[5,3]
```

```
155
```

■ BoedigheimerSeriesQ[lbps,ubps,fphi]

This function will return True if the system (based on the upper and lower boundary points) meets Boedigheimer's [1992] definition of a series system. We use below the lower and upper boundary points that were obtained in the examples for LBPFromStructureand UBPFromStructure. This definition is met if there is one lower boundary point and n upper boundary points to each level.

```
BoedigheimerSeriesQ[lbps,ubps,{0,1,2,3}]
```

```
False
```

■ BoedigheimerParallelQ[lbps,ubps,fphi]

This function will return True if the system (based on the upper and lower boundary points) meets Boedigheimer's [1992] definition of a parallel system. We use below the lower and upper boundary points that were obtained in the examples for LBPFromStructureand UBPFromStructure. This definition is met if there is one upper boundary point and n lower boundary points to each level.

```
BoedigheimerParallelQ[lbps,ubps,{0,1,2,3}]
```

```
False
```

■ LBPFromPaths[paths,n]

This function will return the full lower boundary point form (as is normally returned by functions such as LBPFromStructure)for a list of lists of components, where each list of components identifies a minimal path for a binary system with n components.

```
lbps2=LBPFromPaths[{{1,2,3},{2,4},{5}},6]
```

```
{{{1, 1, 1, 0, 0, 0}, 1, Lower, Real},
 {{0, 1, 0, 1, 0, 0}, 1, Lower, Real},
 {{0, 0, 0, 0, 1, 0}, 1, Lower, Real}}
```

■ UBPFromCuts[cuts,n]

This function will return the full upper boundary point form (as is normally returned by functions such as UBPFromStructure)for a list of lists of components, where each list of components identifies a minimal cut for a binary system with n components.

```
ubps2=UBPFromCuts[{{1,2,3},{2,4},{5}},6]
```

```
{{{0, 0, 0, 1, 1, 1}, 0, Upper, Real},
 {{1, 0, 1, 0, 1, 1}, 0, Upper, Real},
 {{1, 1, 1, 1, 0, 1}, 0, Upper, Real}}
```

■ PathsFromLBP[lbps]

This function will return the list of minimal paths for a binary system, based on the full lower boundary point form (as is normally returned by functions such as LBPFromStructure)for that system..

```
PathsFromLBP[lbps2]
```

```
{{1, 2, 3}, {2, 4}, {5}}
```

■ CutsFromUBP[ubps]

This function will return the list of minimal cuts for a binary system, based on the full upper boundary point form (as is normally returned by functions such as UBPFromStructure)for that system..

```
CutsFromUBP[ubps2]
```

```
{{1, 2, 3}, {2, 4}, {5}}
```

■ BoedigheimerReleventComponentsQ[lbps,ubps,p]

This function will return "True" if every component meets Boedigheimer's [1992] definition of relevency, based on the lower boundary points lbps, the upper boundary points ubps, and the component state space list p.

```
BoedigheimerReleventComponentsQ[lbps,ubps,
   {{0,1,2},{0,1,2,3},{0,1}}]
```

```
True
```

■ StructuralImportances[p,phi]

This function will return the structural importances of every component of the system, based on the component state space list p and the structure function phi.

```
StructuralImportances[{{0,1,2},{0,1,2},{0,1}},phi$Max]
```

$$\left\{\frac{2}{3}, \frac{2}{3}, \frac{1}{9}\right\}$$

■ UBPToLBP[ubps,p,fphi]

This function will return the lower boundary point list, based on the upper boundary point list, the component state space list p, and the system state space fphi. No attempt is made to discern whether any given boundary point is "Real" or "Virtual". However, it should be noted that nowhere in this package does any function actually use this "Real" vs. "Virtual" information.

```
UBPToLBP[ubps,{{0,1,2},{0,1,2,3},{0,1}},{0,1,2,3}]
```

```
{{{0, 0, 1}, 1, Lower, Indet}, {{0, 1, 0}, 1, Lower, Indet},
 {{1, 0, 0}, 1, Lower, Indet}, {{0, 2, 0}, 2, Lower, Indet},
 {{2, 0, 0}, 2, Lower, Indet}, {{0, 3, 0}, 3, Lower, Indet}}
```

■ LBPToUBP[lbps,p,fphi]

This function will return the upper boundary point list, based on the lower boundary point list, the component state space list p, and the system state space fphi. No attempt is made to discern whether any given boundary point is "Real" or "Virtual". However, it should be noted that nowhere in this package does any function actually use this "Real" vs. "Virtual" information, and that the function BPTypeFind is capabale of performing this calculation if it is needed.

```
LBPToUBP[lbps,{{0,1,2},{0,1,2,3},{0,1}},{0,1,2,3}]
```

```
{{{0, 0, 0}, 0, Upper, Indet}, {{1, 1, 1}, 1, Upper, Indet},
 {{2, 2, 1}, 2, Upper, Indet}}
```

■ CutsToPaths[cuts,n]

This function will return the list of minimal paths, based on the list of minimal cuts, for a binary system with n components.

```
CutsToPaths[{{1,2,3,4}},4]
```

```
{{4}, {3}, {2}, {1}}
```

■ PathsToCuts[paths,n]

This function will return the list of minimal cuts, based on the list of minimal paths, for a binary system with n components.

```
CutsToPaths[{{1},{2},{3},{4}},4]
```

```
{{1, 2, 3, 4}}
```

■ SystemStateFromLBP[lbps,fphi,x]

This function will return the state of the system associated with a particular component state vector x, based not on knowledge of the structure function phi but rather on knowledge of the lower boundary point list lbps and the system state space fphi.

```
SystemStateFromLBP[lbps,{0,1,2,3},{0,1,0}]
```

```
1
```

■ SystemStateFromUBP[ubps,fphi,x]

This function will return the state of the system associated with a particular component state vector x, based not on knowledge of the structure function phi but rather on knowledge of the upper boundary point list ubps and the system state space fphi.

```
SystemStateFromUBP[ubps,{0,1,2,3},{0,1,0}]
```

```
1
```

■ BPClean[lbps or ubps,string1,string2]

This function is designed to simplify entry of long lists of boundary points. It will complete some uncompleted fields, and will sort the resulting list. If the upper/lower indication is not present for some boundary point, it will be added as given by string1. If the Real/Virtual indication is not present, it will be added as given by string2. If no string2 is given, it defaults to Indet (for Indeterminate). As a shortcut, this function will also convert codes for the Real/Virtual/Indet entry. It will interpret 0 as Real, 1 as Virtual, and 2 as Indet.

```
simplbps={{{0,0,1},1}, {{0,1,0},1}, {{1,0,0},1},
          {{0,2,0},2}, {{2,0,0},2}, {{0,3,0},3}}
```

```
{{{0, 0, 1}, 1}, {{0, 1, 0}, 1}, {{1, 0, 0}, 1}, {{0, 2, 0}, 2},
 {{2, 0, 0}, 2}, {{0, 3, 0}, 3}}
```

```
BPClean[simplbps,"Lower","Real"]
```

```
{{{0, 0, 1}, 1, Lower, Real}, {{0, 1, 0}, 1, Lower, Real},
 {{1, 0, 0}, 1, Lower, Real}, {{0, 2, 0}, 2, Lower, Real},
 {{2, 0, 0}, 2, Lower, Real}, {{0, 3, 0}, 3, Lower, Real}}
```

■ LBPSelfConsistentQ[lbps]

This function will return "True" if the set of lower boundary points given is consistent with itself, and "False" otherwise.

```
LBPSelfConsistentQ[lbps]
```

```
True
```

■ UBPSelfConsistentQ[ubps]

This function will return "True" if the set of upper boundary points given is consistent with itself, and "False" otherwise.

```
UBPSelfConsistentQ[ubps]
```

```
True
```

■ BPConsistentToEachOtherQ[lbps,ubps]

This function will return "True" if the sets of boundary points given are consistent with each other, and "False" otherwise.

```
BPConsistentToEachOtherQ[lbps,ubps]
```

```
True
```

■ SystemLimitsFromBP[lbps,ubps]

This function will return the extreme states of the system, based on the upper and lower boundary points. The assumption is made that the boundary point sets given are complete and valid.

```
SystemLimitsFromBP[lbps,ubps]
```

```
{0, 3}
```

■ StructureFromPhi[p,phi]

This function will create a table of all possible component states with the associated system states, based only on p and phi.

```
StructureFromPhi[{{0,1,2},{0,1,2,3},{0,1}},phi$Max]
```

```
{{{0, 0, 0}, 0}, {{0, 0, 1}, 1}, {{0, 1, 0}, 1}, {{0, 1, 1}, 1},
 {{0, 2, 0}, 2}, {{0, 2, 1}, 2}, {{0, 3, 0}, 3}, {{0, 3, 1}, 3},
 {{1, 0, 0}, 1}, {{1, 0, 1}, 1}, {{1, 1, 0}, 1}, {{1, 1, 1}, 1},
 {{1, 2, 0}, 2}, {{1, 2, 1}, 2}, {{1, 3, 0}, 3}, {{1, 3, 1}, 3},
 {{2, 0, 0}, 2}, {{2, 0, 1}, 2}, {{2, 1, 0}, 2}, {{2, 1, 1}, 2},
 {{2, 2, 0}, 2}, {{2, 2, 1}, 2}, {{2, 3, 0}, 3}, {{2, 3, 1}, 3}}
```

■ StructureFromLBP[p,lbps,fphi]

This function will create a table of all possible component states with the associated system states, based only on p, fphi and the set of lower boundary points lbps.

```
StructureFromLBP[{{0,1,2},{0,1,2,3},{0,1}},
                 lbps,{0,1,2,3}]
```

```
{{{0, 0, 0}, 0}, {{0, 0, 1}, 1}, {{0, 1, 0}, 1}, {{0, 1, 1}, 1},
 {{0, 2, 0}, 2}, {{0, 2, 1}, 2}, {{0, 3, 0}, 3}, {{0, 3, 1}, 3},
 {{1, 0, 0}, 1}, {{1, 0, 1}, 1}, {{1, 1, 0}, 1}, {{1, 1, 1}, 1},
 {{1, 2, 0}, 2}, {{1, 2, 1}, 2}, {{1, 3, 0}, 3}, {{1, 3, 1}, 3},
 {{2, 0, 0}, 2}, {{2, 0, 1}, 2}, {{2, 1, 0}, 2}, {{2, 1, 1}, 2},
 {{2, 2, 0}, 2}, {{2, 2, 1}, 2}, {{2, 3, 0}, 3}, {{2, 3, 1}, 3}}
```

■ StructureFromUBP[p,ubps,fphi]

This function will create a table of all possible component states with the associated system states, based only on p, fphi and the set of upper boundary points ubps.

```
StructureFromUBP[{{0,1,2},{0,1,2,3},{0,1}},
                 ubps,{0,1,2,3}]
```

```
{{{0, 0, 0}, 0}, {{0, 0, 1}, 1}, {{0, 1, 0}, 1}, {{0, 1, 1}, 1},
 {{0, 2, 0}, 2}, {{0, 2, 1}, 2}, {{0, 3, 0}, 3}, {{0, 3, 1}, 3},
 {{1, 0, 0}, 1}, {{1, 0, 1}, 1}, {{1, 1, 0}, 1}, {{1, 1, 1}, 1},
 {{1, 2, 0}, 2}, {{1, 2, 1}, 2}, {{1, 3, 0}, 3}, {{1, 3, 1}, 3},
 {{2, 0, 0}, 2}, {{2, 0, 1}, 2}, {{2, 1, 0}, 2}, {{2, 1, 1}, 2},
 {{2, 2, 0}, 2}, {{2, 2, 1}, 2}, {{2, 3, 0}, 3}, {{2, 3, 1}, 3}}
```

■ SystemSpaceFromBP[lbps,ubps]

This function will discern the system state space fphi based on the complete sets of upper and lower boundary points for the system.

```
SystemSpaceFromBP[lbps,ubps]
```

```
{0, 1, 2, 3}
```

■ BPTypeFind[bplist,phi]

This function will discern whether each boundary point is either Real or Virtual, converting any Indet values in the table to one of those two designations. The structure function phi is necessary input.

```
lbps3 = {{{0, 0, 2}, 1, "Lower", "Indet"},
         {{0, 2, 0}, 1, "Lower", "Indet"},
         {{2, 0, 0}, 1, "Lower", "Indet"},
         {{0, 0, 2}, 2, "Lower", "Indet"},
         {{0, 2, 0}, 2, "Lower", "Indet"},
         {{2, 0, 0}, 2, "Lower", "Indet"},
         {{0, 3, 0}, 3, "Lower", "Indet"}}
```

```
{{{0, 0, 2}, 1, Lower, Indet}, {{0, 2, 0}, 1, Lower, Indet},
 {{2, 0, 0}, 1, Lower, Indet}, {{0, 0, 2}, 2, Lower, Indet},
 {{0, 2, 0}, 2, Lower, Indet}, {{2, 0, 0}, 2, Lower, Indet},
 {{0, 3, 0}, 3, Lower, Indet}}
```

```
BPTypeFind[lbps3,phi$Max]
```

```
{{{0, 0, 2}, 1, Lower, Virtual}, {{0, 2, 0}, 1, Lower, Virtual},
 {{2, 0, 0}, 1, Lower, Virtual}, {{0, 0, 2}, 2, Lower, Real},
 {{0, 2, 0}, 2, Lower, Real}, {{2, 0, 0}, 2, Lower, Real},
 {{0, 3, 0}, 3, Lower, Real}}
```

The StochasticAnalysisPackage

■ General Comments and Definitions

We start by defining a multistate system that we will use throughout this chapter for the sake of example. As is generally the case, "p" is the list of possible state values for each component of the system, "fphi" is the list of possible state values for the system itself, and "pprob" is the list of probabilities that each component of the system is in each one of its given states. phi$Enum will be a structure function which was explicitly defined by the customer, and phi$Max is a structure function which assumes the maximal value of any of the components

```
p={Range[0,3],Range[0,2]}
```

```
{{0, 1, 2, 3}, {0, 1, 2}}
```

```
pprob={{0.05, 0.1, 0.15, 0.7},{0.1, 0.3, 0.6}}
```

```
{{0.05, 0.1, 0.15, 0.7}, {0.1, 0.3, 0.6}}
```

```
fphi=Range[0,4]
```

```
{0, 1, 2, 3, 4}
```

```
phi$Max[x_] := Max[x]
```

```
phi$Enum[x_] := Module[{systab},
   systab = {{{0,0},0},
             {{0,1},1},
             {{0,2},2},
             {{1,0},0},
             {{1,1},1},
             {{1,2},3},
             {{2,0},1},
             {{2,1},2},
             {{2,2},3},
             {{3,0},2},
             {{3,1},4},
```

As it will come in handy later, we now calculate the lower boundary points for this system:

```
(lbps=LBPFromStructure[p,phi$Enum,fphi]) // MatrixForm
```

{0, 1}	1	Lower	Real
{2, 0}	1	Lower	Real
{0, 2}	2	Lower	Real
{2, 1}	2	Lower	Real
{3, 0}	2	Lower	Real
{1, 2}	3	Lower	Real
{3, 1}	3	Lower	Virtual
{3, 1}	4	Lower	Real

■ Function Documentation

■ ConsistentProbabilitiesQ[p, pprob]

This function will check that the matrix of component state probabilities is the same size as the matrix of component state values, and that the probability that each component is in SOME state is one. It operates only with discrete systems.

```
ConsistentProbabilitiesQ[p,pprob]
```

```
True
```

■ SystemFromDirectEnumeration[pphi, fphi, pprob]

This function will return the exact system state probabilities for a discrete system, based on the component state spaces p, the system state space fphi, the probabilities of each component being in each state pprob, and the structure function phi.

```
SystemFromDirectEnumeration[p,phi$Enum,fphi,pprob]
```

```
{0.015, 0.06, 0.145, 0.15, 0.63}
```

■ SystemFromLBPInclusionExclusion[p, lbps, fphi, pprob]

This function will return the exact system state probabilities for a discrete system. In contrast to SystemFromDirectEnumeration, this function attempts to use Feller's Inclusion/Exclusion Principle to save on computation time.

```
SystemFromLBPInclusionExclusion[p,lbps,fphi,pprob]
```

```
{0.015, 0.06, 0.145, 0.15, 0.63}
```

■ TrivialBoundsFromLBP[p,lbpoint, pprob]

This function will return the trivial bounds for the probability of being in or above the particular state of the discrete system which corresponds to the lower boundary point which was given. Note that only one lower boundary point is given, as a simple list of the component states (i.e. {2,2,1}. The function returns a list of two numbers, where the first is the lower bound and the second is the upper bound.

```
TrivialBoundsFromLBP[p, {1,2}, pprob]
```

```
{0.57, 0.98}
```

■ InclusionExclusionBoundsFromLBP[p,lbps, fphi, pprob, prec]

This function will return the Inclusion/Exclusion bounds on the probability of being in or above any particular state of the discrete system. prec is the maximum number of Inclusion/Exclusion terms to use in the summation (the lower this number, the rougher and faster the approximation). Note that, if, for any state, the number of summation terms needed to exactly compute the probability in question is less than or equal to prec, then an exact value for that state is returned rather than bounds.

```
InclusionExclusionBoundsFromLBP[p,lbps,fphi,pprob,2]
```

```
{1, 0.985, {0.505, 2.065}, 0.78, 0.63}
```

■ SystemMatrix[fphi,ans]

This function will merge the system system states with the system probabilities, for further examination in terms of reliability measures. It functions only with discrete systems

```
SystemMatrix[fphi,
  SystemFromDirectEnumeration[p,phi$Enum,fphi,pprob]]
```

```
{{0, 0.015}, {1, 0.06}, {2, 0.145}, {3, 0.15}, {4, 0.63}}
```

■ ReliabilityImportance[p,phi, fphi, pprob, i, j]

This function will return the Reliability Importance of component i in state j in the given system, for discrete systems.

```
ReliabilityImportance[p, phi$Enum, fphi, pprob, 1, 1]
```

```
0.6
```

■ ReliabilityImportancesTable[p,phi, fphi, pprob]

This function will return the Reliability Importances for every state of every component. This is returned in the same form as p, and functions only for discrete systems.

```
ReliabilityImportancesTable[p, phi$Enum, fphi, pprob]
```

```
{{0., 0.6, 1., 2.3}, {0., 1.7, 2.1}}
```

■ PToQ[pr]

This function will accept a vector of probabilities of being in a particular state and return the probabilities of being over and above each particular state. It has meaning only for discrete systems.

```
PToQ[SystemFromDirectEnumeration[p,phi$Enum,
                                 fphi,pprob]]
```

```
{1, 0.985, 0.925, 0.78, 0.63}
```

■ QToP[qr]

This function will accept a vector of probabilities of being in or above any particular state and return the probabilities of being in each particular state. It has meaning only for discrete systems.

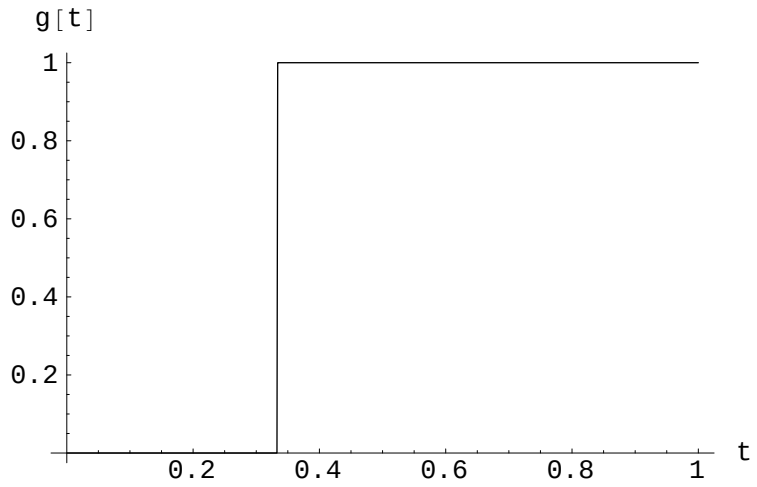
```
QToP[PToQ[SystemFromDirectEnumeration[p,phi$Enum,
                                      fphi,pprob]]]
```

```
{0.015, 0.06, 0.145, 0.15, 0.63}
```

■ StieltjesIntegral[f,g, {t, min:0, max:1}, fact:2]

This function will return an approximationto the Stieltjes integral of f with respect to function g, where f and g are functions of t. The integration takes places using 10^fact+1 samplings of f and g between min and max. f and/or g may be discrete or continuous.

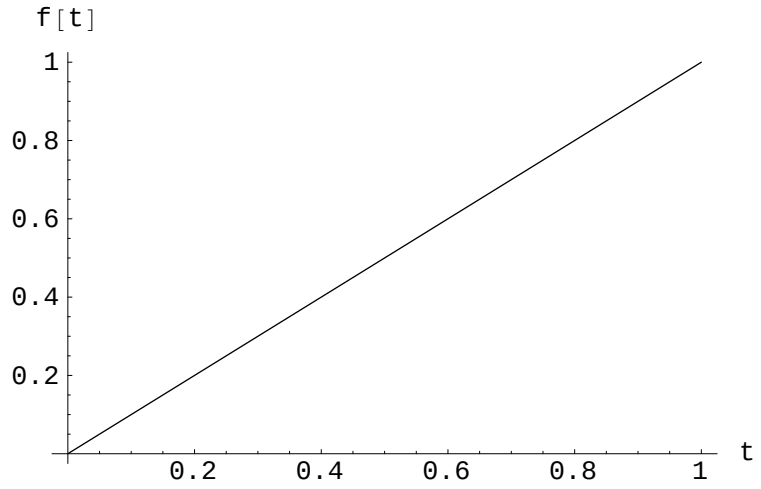
```
g[t_] := Which[t<(1/3), 0,
               t>=(1/3),1]

Plot[g[t], {t,0,1}, AxesLabel->{"t","g[t]"}]
```




```
- Graphics -
```

```
f[t_] := t
```

```
Plot[f[t], {t,0,1}, AxesLabel->{"t","f[t]"}]
```




```
- Graphics -
```

```
StieltjesIntegral[f[t], g[t], {t, 0, 1}, 2]
```

```
0.34
```

■ StieltjesIntegralH[f,g, {t, min:0, max:1}, fact:2]

This function will return an approximationto the Stieltjes integral of f with respect to function g, where f and g are functions of t. The integration takes places using 10^fact+1 samplings of f and g between min and max, and then uses the NSum function to estimate the contribution to the integral of regions between max and Infinity. See StieltjesIntegral for definitions of f and g that are used in this example. f and/or g may be discrete or continuous.

```
StieltjesIntegralH[f[t], g[t], {t, 0, 1}, 2]
```

```
0.34
```

■ StieltjesIntegralG[f,g, {t, min:0, max:1}, fact:2]

This function will return an approximationto the Stieltjes integral of f with respect to function g, where f and g are functions of t. The integration takes places using 10^fact+1 samplings of f and g between min and max, and then uses the NSum function to estimate the contribution to the integral of regions between max and Infinity and between -Infinity and min. See StieltjesIntegral for definitions of f and g that are used in this example. f and/or g may be discrete or continuous.

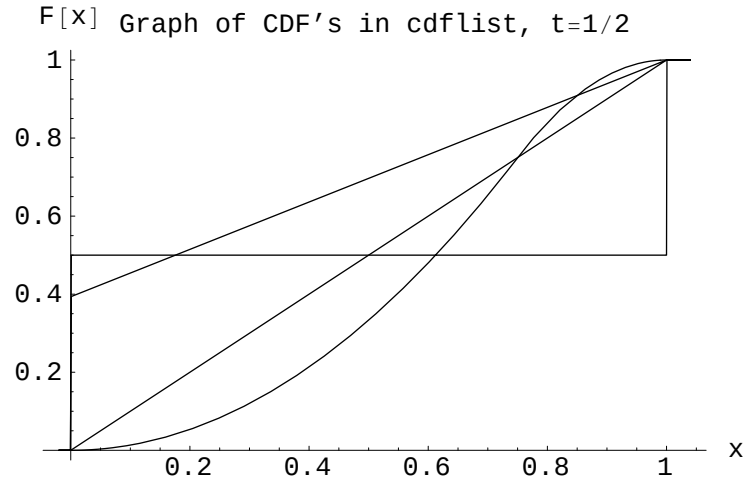
```
StieltjesIntegralG[f[t], g[t], {t, 0, 1}, 2]
```

```
0.34
```

■ P[cdfs, rules:none]

This function will return the CDF of a random variable which is a maximum of those given in the list named cdfs through their cumulative distribution functions. If you wish not the general solution, but rather an answer for a particular x, then enter that as an optional rule such as x->0.5. If you wish an answer for a particular moment in time t, then you can enter that as x->x0, t->t0. It is important to be clear on the meaning of the CDF for a point in time for a component. This CDF is a function that, at that point in time, gives the probability of that component being in or below any given state. It has the same meaning and functional form for binary, multistate, continuous, and mixed components.

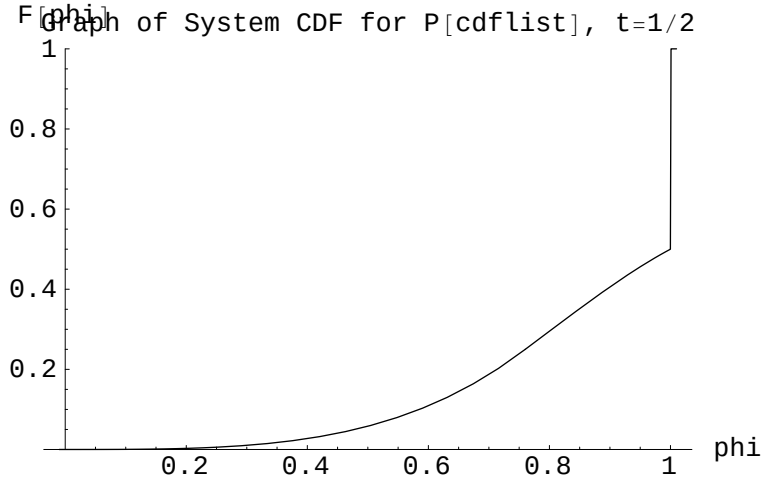
```
cdflist={BinaryCDF[x,1/2],TriangularCDF[x,3/4],
        UniformCDF[x],UniformMixedCDF[x,1-E^(-t),0]};

Plot[Evaluate[cdflist /. t->(1/2)], {x, -0.02, 1.04},
   PlotLabel->"Graph of CDF's in cdflist, t=1/2",
   AxesLabel->{"x","F[x]"}]
```




```
- Graphics -
```

```
psystem=P[cdflist, t->(1/2)];
```

```
Plot[psystem, {x, -0.01, 1.01},
   PlotLabel->
      "Graph of System CDF for P[cdflist], t=1/2",
   AxesLabel->
      {"phi","F[phi]"},PlotRange->{0,1}]
```



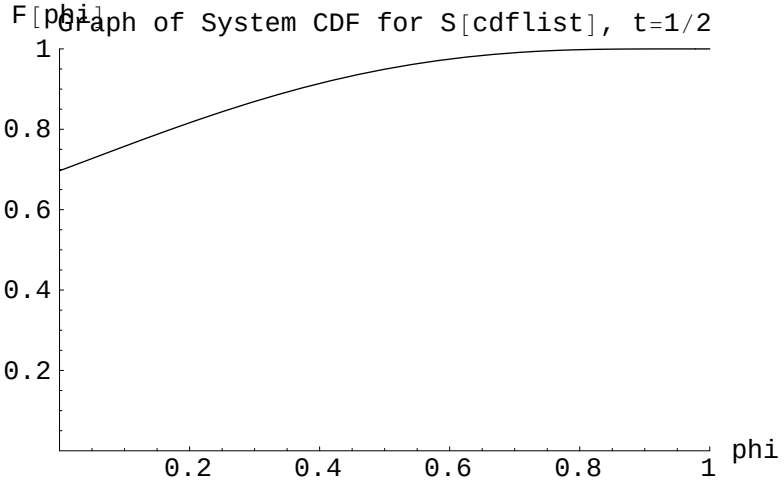

```
- Graphics -
```

■ S[cdfs, rules:none]

This function will return the CDF of a random variable which is a minimum of those given in the list named cdfs through their cumulative distribution functions. If you wish not the general solution, but rather an answer for a particular x, then enter that as an optional rule such as x->0.5. If you wish an answer for a particular moment in time t, then you can enter that as x->x0, t->t0. It is important to be clear on the meaning of the CDF for a point in time for a component. This CDF is a function that, at that point in time, gives the probability of that component being in or below any given state. It has the same meaning and functional form for binary, multistate, continuous, and mixed components. Please see P for the definition of the CDF list used in the illustration of this function.

```
ssystem=S[cdflist, t->(1/2)];
```

```
Plot[ssystem, {x, 0, 1},
   PlotLabel->
      "Graph of System CDF for S[cdflist], t=1/2",
   AxesLabel->{"phi","F[phi]"},
   PlotRange->{{0,1},{0,1}}]
```



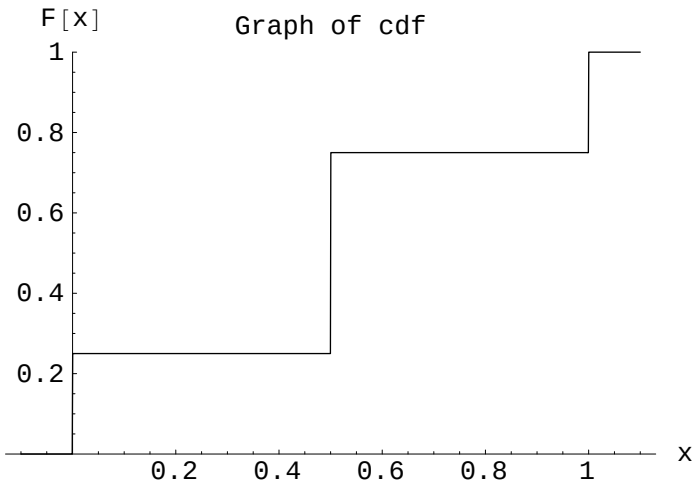

```
- Graphics -
```

■ A[cdf, x, a]

This function will return the CDF of a random variable y where y=ax. a is some real number, and x is another random variable. This has the same meaning and form for continuous, multistate, binary, and mixed systems.

```
cdf=MultistateCDF[x,{{0,1/4},{1/2,1/2},{1,1/4}}];
```

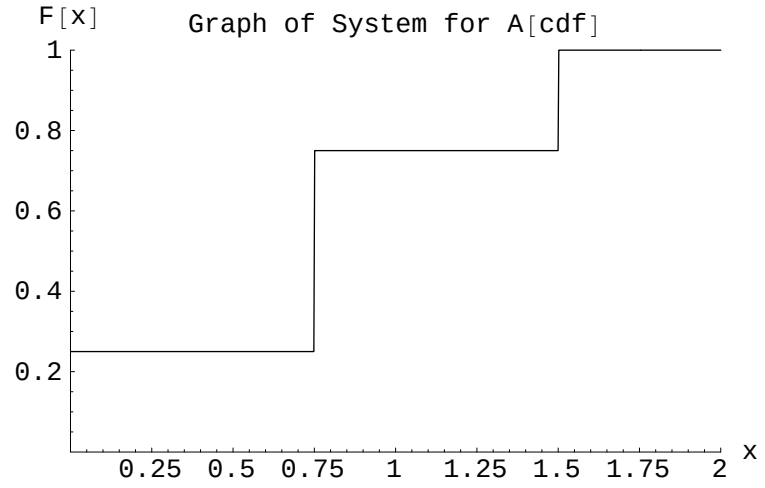
```
Plot[cdf // Evaluate, {x, -0.1, 1.1},
     PlotLabel->"Graph of cdf",
     AxesLabel->{"x","F[x]"},
     PlotRange->{0,1}]
```



- Graphics -

```
asystem=A[cdf, x, 3/2];
```

```
Plot[asystem // Evaluate, {x, 0, 2},
   PlotLabel->"Graph of System for A[cdf]",
   AxesLabel->{"x","F[x]"},
   PlotRange->{{0,2},{0,1}}]
```



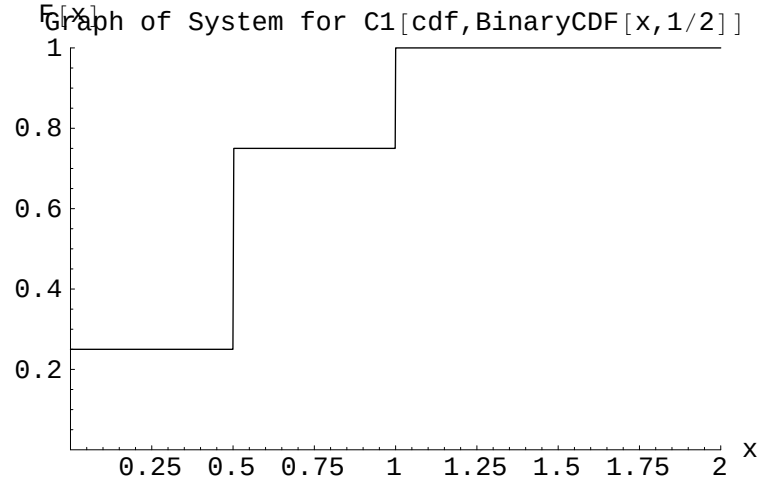

```
- Graphics -
```

■ C1[cdf1, {cdf2, min2:0, max2:1}, x, z, fact:2]

This function will return the probability that x+y is less than or equal to some value z. The CDF of x is given by cdf1, and the CDF of y is given by cdf2. min2 and max2 are the minimum and maximum values that the random variable y may assume (note: min and max parameters are common for many types of functions in RelPack, and have generally the same interpretation - their presence will not always be commented on in the documentation for the function, if they are identifiable in the parameter list for that function). It is assumed that cdf1 and cdf2 are functions of x. fact has the same interpretation that is does in StieltjesIntegral. If z is not given as a real value, then the CDF is returned to you as a function of x.

```
c1system=C1[cdf // Hold, {BinaryCDF[x,1/2]}, x, z];
```

```
Plot[c1system // ReleaseHold // Evaluate, {x, 0, 2},
 PlotLabel->
  "Graph of System for C1[cdf,BinaryCDF[x,1/2]]",
 AxesLabel->{"x","F[x]"},
 PlotRange->{{0,2},{0,1}}]
```



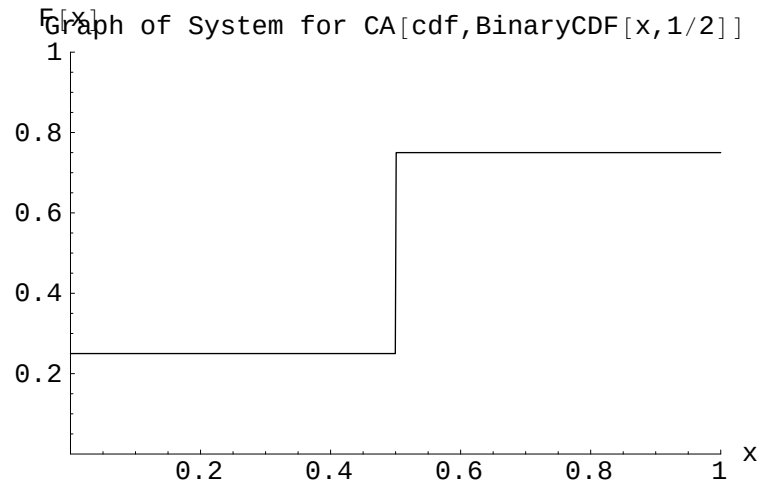

```
- Graphics -
```

■ CA[cdf1, min1:0, max:1, {cdf2, min2:0, max2:1}, x, z, fact:2]

This function is identical to C1, except that the result is scaled to have a maximum value of 1.

```
casystem=CA[cdf // Hold, {BinaryCDF[x,1/2]}, x, z];
```

```
Plot[casystem // ReleaseHold // Evaluate, {x, 0, 1},
 PlotLabel->
  "Graph of System for CA[cdf,BinaryCDF[x,1/2]]",
 AxesLabel->{"x","F[x]"}, PlotRange->{{0,1},{0,1}}]
```



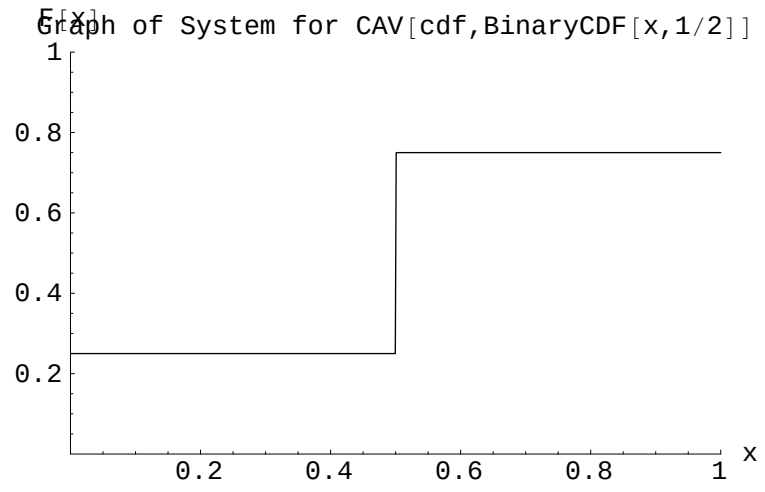

```
- Graphics -
```

■ CAV[cdf1, min1:0, max:1, {cdf2, min2:0, max2:1}, x, z, fact:2]

This function is identical to C1, except that the CDF for the average, rather than the sum, of the random variables represented by cdf1 and cdf2 is returned. When min1=min2=0 and max1=max2=1, this will return the same result as CA.

```
cavsystem=CAV[cdf // Hold, {BinaryCDF[x,1/2]}, x, z];
```

```
Plot[cavsystem // ReleaseHold // Evaluate, {x, 0, 1},
 PlotLabel->
  "Graph of System for CAV[cdf,BinaryCDF[x,1/2]]",
 AxesLabel->{"x","F[x]"}, PlotRange->{{0,1},{0,1}}]
```




```
- Graphics -
```

■ DiscreteBuild[syslist,phi]

This function will take a list of complete sys matrices for different subsystems, and apply phi to them so as to construct a sys matrix for the whole system. The probability of the system being in any one of its possible states is determined by direct enumeration.

```
x1={{0,1/4},{1/2,1/2},{1,1/4}};

x2={{0,1/2},{1,1/2}};

x3={{0,1/3},{1/2,1/3},{1,1/3}};

syslist={x1,x2,x3};
```

```
DiscreteBuild[syslist,phi$Max]
```

$$\left\{\left\{0, \frac{1}{24}\right\}, \left\{\frac{1}{2}, \frac{5}{24}\right\}, \left\{1, \frac{3}{4}\right\}\right\}$$

■ PM[syslist]

This function is the purely discrete analog of P. Note that P will still work properly with discrete systems, but if you know ahead ahead of time that the entire system is completely discrete, you will save time by using PM rather than P. syslist is in the same form as it was given in DiscreteBuild.

```
PM[syslist]
```

$$\left\{\left\{0, \frac{1}{24}\right\}, \left\{\frac{1}{2}, \frac{5}{24}\right\}, \left\{1, \frac{3}{4}\right\}\right\}$$

■ SM[syslist]

This function is the purely discrete analog of S. See PM for details.

```
SM[syslist]
```

$$\left\{\left\{0, \frac{3}{4}\right\}, \left\{\frac{1}{2}, \frac{5}{24}\right\}, \left\{1, \frac{1}{24}\right\}\right\}$$

■ AM[sys, a]

This function is the purely discrete analog of A. See PM for details. Note that unlike syslist, here we are passing only one system matrix.

```
AM[syslist[[2]], 3/2]
```

$$\left\{\left\{0, \frac{1}{2}\right\}, \left\{\frac{3}{2}, \frac{1}{2}\right\}\right\}$$

■ C1M[syslist]

This function is the purely discrete analog of C1. See PM for details.

C1M[syslist]

$$\left\{\left\{0, \frac{1}{24}\right\}, \left\{\frac{1}{2}, \frac{1}{8}\right\}, \left\{1, \frac{5}{24}\right\}, \left\{\frac{3}{2}, \frac{1}{4}\right\}, \left\{2, \frac{5}{24}\right\}, \left\{\frac{5}{2}, \frac{1}{8}\right\}, \left\{3, \frac{1}{24}\right\}\right\}$$

■ CAM[syslist]

This function is the purely discrete analog of CA. See PM for details.

CAM[syslist]

$$\left\{\left\{0, \frac{1}{24}\right\}, \left\{\frac{1}{6}, \frac{1}{8}\right\}, \left\{\frac{1}{3}, \frac{5}{24}\right\}, \left\{\frac{1}{2}, \frac{1}{4}\right\}, \left\{\frac{2}{3}, \frac{5}{24}\right\}, \left\{\frac{5}{6}, \frac{1}{8}\right\}, \left\{1, \frac{1}{24}\right\}\right\}$$

■ CAVM[syslist]

This function is the purely discrete analog of CAV. See PM for details.

CAVM[syslist]

$$\left\{\left\{0, \frac{1}{24}\right\}, \left\{\frac{1}{6}, \frac{1}{8}\right\}, \left\{\frac{1}{3}, \frac{5}{24}\right\}, \left\{\frac{1}{2}, \frac{1}{4}\right\}, \left\{\frac{2}{3}, \frac{5}{24}\right\}, \left\{\frac{5}{6}, \frac{1}{8}\right\}, \left\{1, \frac{1}{24}\right\}\right\}$$

■ Additional Commentary

■ Comments on Timing

In many circumstances, SystemFromLBPInclusionExclusion and InclusionExclusionBoundsFromLBP will perform their tasks faster than SystemFromDirectEnumeration. The following example will illustrate this for phi$Max.

■ Definitions

p$1=Table[Range[0,3],{4}]

```
{{0, 1, 2, 3}, {0, 1, 2, 3}, {0, 1, 2, 3}, {0, 1, 2, 3}}
```

```
fphi$1=Range[0,3]
```

```
{0, 1, 2, 3}
```

```
pprob$1 = {{0.5043128477021241, 0.07245055618411537,
            0.05287763869180079, 0.3703589574219598},
           {0.087617398287609, 0.3289199767039768,
            0.2411610389233241, 0.3423015860850898},
           {0.4205428683153921, 0.06576473583833683,
            0.407974829002557, 0.105717566843714},
           {0.2303276367256573, 0.2358913729230182,
            0.3206308214277661, 0.2131501689235582}};
```

```
lbps$1=LBPFromStructure[p$1,phi$Max,fphi$1]
```

```
{{{0, 0, 0, 1}, 1, Lower, Real}, {{0, 0, 1, 0}, 1, Lower, Real},
 {{0, 1, 0, 0}, 1, Lower, Real}, {{1, 0, 0, 0}, 1, Lower, Real},
 {{0, 0, 0, 2}, 2, Lower, Real}, {{0, 0, 2, 0}, 2, Lower, Real},
 {{0, 2, 0, 0}, 2, Lower, Real}, {{2, 0, 0, 0}, 2, Lower, Real},
 {{0, 0, 0, 3}, 3, Lower, Real}, {{0, 0, 3, 0}, 3, Lower, Real},
 {{0, 3, 0, 0}, 3, Lower, Real}, {{3, 0, 0, 0}, 3, Lower, Real}}
```

■ Calculations

```
Timing[PToQ[SystemFromDirectEnumeration[p$1,
   phi$Max,fphi$1,pprob$1]]]
```

```
{0.711 Second, {1, 0.99572, 0.945531, 0.708602}}
```

```
Timing[PToQ[SystemFromLBPInclusionExclusion[p$1,
   lbps$1,fphi$1,pprob$1]]]
```

```
{0.12 Second, {1, 0.99572, 0.945531, 0.708602}}
```

```
Timing[InclusionExclusionBoundsFromLBP[p$1,
   lbps$1,fphi$1,pprob$1,3]]
```

```
{0.1 Second, {1, {-0.0407177, 1.19742}, {0.47854, 1.01324},
  {0.654976, 0.711459}}}
```

```
Timing[InclusionExclusionBoundsFromLBP[p$1,
   lbps$1,fphi$1,pprob$1,2]]
```

```
{0.08 Second,
 {1, {-0.0407177, 2.7572}, {0.47854, 2.05417}, {0.654976, 1.03153}}}
```

■ Interpretation

Note that, in this case, obtaining the system state probabilities from direct enumeration was the most time consuming, obtaining this information from the lower boundary points was faster, and obtaining only bounds on the system state values was faster yet. The less the precision of the bounds desired, the faster the calculation.

The DynamicModels Package

■ General Comments and Definitions

In general, these function use finite-state, continuous time Markov chain methods to perform dynamic modelling for discrete systems. They are of little or no value for continuous or mixed components or systems. Functions beginning with the letters "PDP" are essentially assuming a non-repairable system that begins in its maximal state. Functions beginning with the letters "CTMC" are for general continuous-time, finite-state Markov chains, and may represent repairable behavior. Development of the CTMC functions follows the discussion in *System Reliability Theory* by Arnljot Hoyland and Marvin Rausand

■ Function Documentation

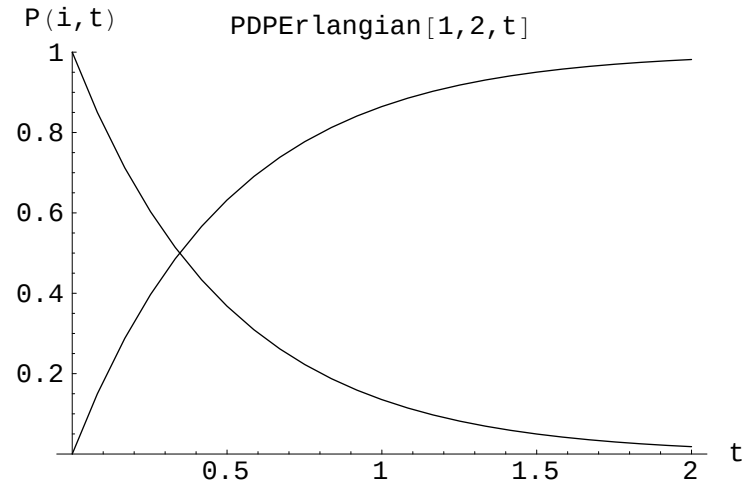
■ PDPErlangian[M,val, t]

This function will return the p(i,t) values for a pure death process where the transition rates are all the same and the system may transition only to the next lower state. val is the value of the common transition rate, and t is the moment in time the solution is desired for. val and/or t may be symbolic or constant; constant values will produce a simplified solution. M must be a positive integer, however. What is returned is a list containing the probability (as a function of t) of being in any of the M+1 states, starting with the lowest state and progressing to the highest (best) state. If one has a fphi vector of length M+1, it can be merged with the result of this calculation to form a complete system matrix, using SystemMatrix[fphi,ans]. It is assumed that the system begins in its maximal state.

```
pdpe1=PDPErlangian[1, L, t]
```

$$\{1 - E^{-L\,t},\ E^{-L\,t}\}$$

```
Plot[Evaluate[pdpe1 /. L->2], {t, 0, 2},
   PlotLabel->"PDPErlangian[1,2,t]",
   AxesLabel->{"t","P(i,t)"},
   PlotRange->{0,1}]
```

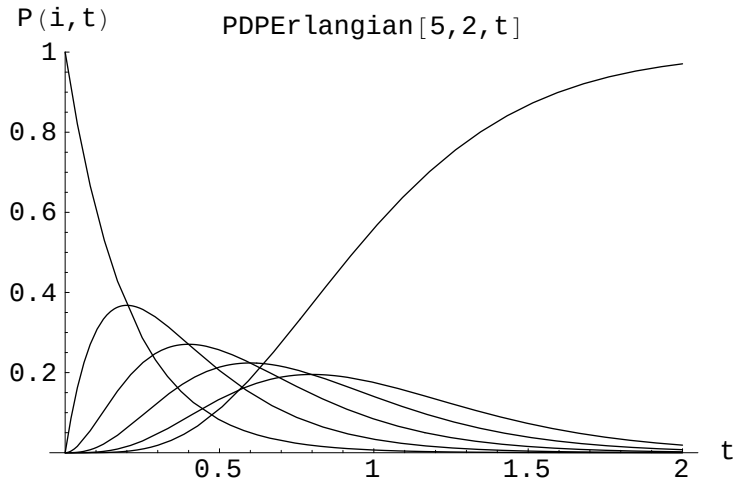


- Graphics -

```
pdpe5 = PDPErlangian[5, L, t]
```

$$\left\{1 - E^{-L\,t} - E^{-L\,t}\,L\,t - \frac{1}{2}\,E^{-L\,t}\,L^2\,t^2 - \frac{1}{6}\,E^{-L\,t}\,L^3\,t^3 - \frac{1}{24}\,E^{-L\,t}\,L^4\,t^4,\right.$$
$$\left.\frac{1}{24}\,E^{-L\,t}\,L^4\,t^4,\ \frac{1}{6}\,E^{-L\,t}\,L^3\,t^3,\ \frac{1}{2}\,E^{-L\,t}\,L^2\,t^2,\ E^{-L\,t}\,L\,t,\ E^{-L\,t}\right\}$$

```
Plot[Evaluate[pdpe5 /. L->5], {t, 0, 2},
   PlotLabel->"PDPErlangian[5,2,t]",
   AxesLabel->{"t","P(i,t)"},
   PlotRange->{0,1}]
```




```
- Graphics -
```

As was the case with the previous graph for this function, the curve which begins as P(i,0)=1 is the probability of being in the maximal state, as a function of time. The curve which approaches 1 as t->Infinity is the probability of being in the minimal state as a function of time. The other curves are intermediary states.

```
PDPErlangian[5, 2, 4] // N
```

```
{0.900368 , 0.0572523 , 0.0286261 , 0.0107348 , 0.0026837 , 0.000335463 }
```

Exact values may be obtained immediately, if symbolic solutions are not desired.

```
Plus @@ PDPErlangian[10, L, t]
```

```
1
```

Note that the sum of the probabilities of being in any given state is 1.

```
SystemMatrix[Range[0,3],PDPErlangian[3, L, t]]
```

$$\left\{\left\{0,\ 1 - E^{-L\,t} - E^{-L\,t}\,L\,t - \frac{1}{2}\,E^{-L\,t}\,L^2\,t^2\right\},\ \left\{1,\ \frac{1}{2}\,E^{-L\,t}\,L^2\,t^2\right\},\ \{2,\ E^{-L\,t}\,L\,t\},\ \{3,\ E^{-L\,t}\}\right\}$$

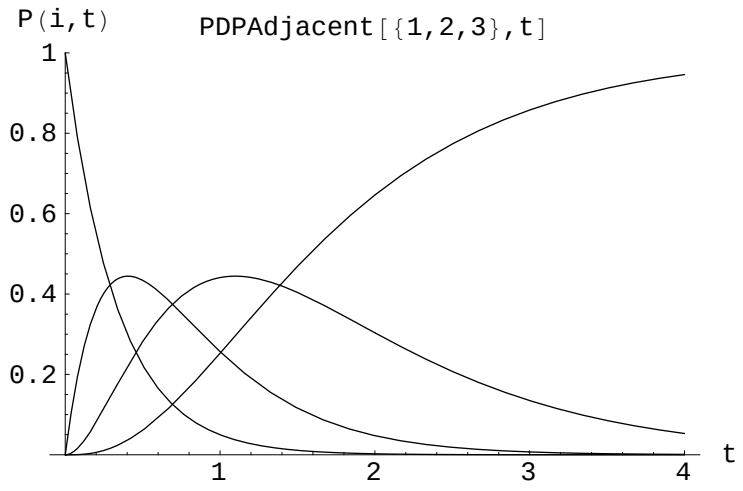
■ PDPAdjacent[mulist,t]

This function will return the p(i,t) values for a pure death process which begins in the maximal state and which may make transitions only to the next lower state. mulist is a list consisting of {m10, m21, ..., mM,M-1}, where mij is the transition rate between state i and state j. Note that for a system of this type there will be one more system state than there are elements in mulist. It is assumed that the elements of mulist are all different. If you give real values for the mulist elements, the calculation will be greatly simplified. The data is returned in the same form they are for PDPErlangian.

```
pdpatest =  PDPAdjacent[{L1,L2,L3},t]
```

$$\left\{L1\,L2\,L3\left(\frac{1}{L1\,L2\,L3} - \frac{E^{-L3\,t}}{(L1-L3)\,(L2-L3)\,L3} - \frac{E^{-L1\,t}}{L1\,(-L1+L2)\,(-L1+L3)} - \frac{E^{-L2\,t}}{(L1-L2)\,L2\,(-L2+L3)}\right),\ L2\,L3\left(\frac{E^{-L3\,t}}{(L1-L3)\,(L2-L3)} + \frac{E^{-L1\,t}}{(-L1+L2)\,(-L1+L3)} + \frac{E^{-L2\,t}}{(L1-L2)\,(-L2+L3)}\right),\ L3\left(\frac{E^{-L3\,t}}{L2-L3} + \frac{E^{-L2\,t}}{-L2+L3}\right),\ E^{-L3\,t}\right\}$$

```
Plot[Evaluate[pdpatest /. {L1->1, L2->2, L3->3}],
      {t, 0, 4},
   PlotLabel->"PDPAdjacent[{1,2,3},t]",
   AxesLabel->{"t","P(i,t)"},
   PlotRange->{0,1}]
```




```
- Graphics -
```

The curve which begins as P(i,0)=1 is the probability of being in the maximal state, as a function of time. The curve which approaches 1 as t->Infinity is the probability of being in the minimal state as a function of time. The other curves are intermediary states.

```
PDPAdjacent[{1,2,3},1] // N
```

```
{0.25258, 0.440988, 0.256645, 0.0497871}
```

```
(PDPAdjacent[{L1,L2,L3},t] /.
    {L1->1, L2->2, L3->3, t->1}) ==
  PDPAdjacent[{1,2,3},1]
```

```
True
```

Numerical results may of course also be obtained, as was the case for the previous function.

■ PDPNonAdjacent[M,mu,t]

This function returns the general form of a pure-death process' sp(i,t) expressions when it is assumed that the system may transition to ANY lower state. As this calculation is entirely symbolic it can take large amounts of time to calculate and produce quite complicated ouput. The transition rates in this function are assumed to be distinct. mu is the variable name which will form the basic tag to which subscripts will be attached to identify transisition rates; it may not be a real number, but must be a symbol. In general, mu[i,j] is the transition rate from state i to state j, where M is the maximal state and 0 is the minimal state. The parametert (for time) may be symbolic, or a real number, as desired. If anything is known about the system transition matrix (such as the real values for the transition rates), CTMCStateProbabilities[] will produce a much faster solution and should be used instead. It is assumed here that the system has M+1 states. For PDPNonAdjacent, M has a maximum value of 19.

```
PDPNonAdjacent[2,L,t]
```

$$\left\{L[2,0]\left(\frac{E^{-t\,(L[2,0]+L[2,1])}}{-L[2,0]-L[2,1]}+\frac{1}{L[2,0]+L[2,1]}\right)+L[1,0]\,L[2,1]\right.$$
$$\left(\frac{E^{-t\,L[1,0]}}{L[1,0]\,(L[1,0]-L[2,0]-L[2,1])}+\frac{1}{L[1,0]\,(L[2,0]+L[2,1])}+\right.$$
$$\left.\frac{E^{-t\,(L[2,0]+L[2,1])}}{(L[2,0]+L[2,1])\,(-L[1,0]+L[2,0]+L[2,1])}\right),$$
$$L[2,1]\left(\frac{E^{-t\,(L[2,0]+L[2,1])}}{L[1,0]-L[2,0]-L[2,1]}+\frac{E^{-t\,L[1,0]}}{-L[1,0]+L[2,0]+L[2,1]}\right),$$
$$\left.E^{-t\,(L[2,0]+L[2,1])}\right\}$$

■ CTMCStateProbabilities[matrix,t,initialstate:maximal]

This function will fine the expressions for the p(i,t) values for a continuous-time,finite-state Markov chain which begins in state initialstate (one of 0, 1, 2, ..., M). This function may be used to represent repairable systems, and represents the most general solution to this type of problem (PDPErlangian[],PDPAdjacent[],and PDPNonAdjacent[]are all special cases of this technique). If initialstate is not specified, then the maximal state is assumed. The parameter for time is assumed to be t. The parameter matrix is the state transisition matrix. This matrix is such that the current state is the column number-1 (with 0 on the extreme left) and the next state is the row number-1 (with 0 on the top). As with all matrix matrices that are used as input in this package, the diagonal elements of the transition matrix (i.e. the rate of transition between each state and itself) does not need to be specified (and in this developmentmay be replaced by 0 values, if desired). The form of the solution is the same as that for all the "PDP" functions (a list of probabilities of the system being in each state, from the minimal state to the maximal state, as a function of time). t may of course be either symbolic or numerical. matrix may also be symbolic or numerical, although answers will certain be returned more rapidly if matrix is numerical.

```
(sampmat1  = {{0,  lambda},
              {mu,       0}}) // MatrixForm
```

$$\begin{pmatrix} 0 & \text{lambda} \\ \text{mu} & 0 \end{pmatrix}$$

```
CTMCStateProbabilities[sampmat1,t]
```

$$\left\{\text{lambda}\left(\frac{E^{-(\text{lambda}+\text{mu})\,t}}{-\text{lambda}-\text{mu}}+\frac{1}{\text{lambda}+\text{mu}}\right),\right.$$
$$\left.\frac{E^{-(\text{lambda}+\text{mu})\,t}\,\text{lambda}}{\text{lambda}+\text{mu}}+\frac{\text{mu}}{\text{lambda}+\text{mu}}\right\}$$

Note that this was a two-state repairable system, with the transition rate from the maximal state to the minimal state being lambda, and the transition rate from the minimal state to the maximal state being mu.

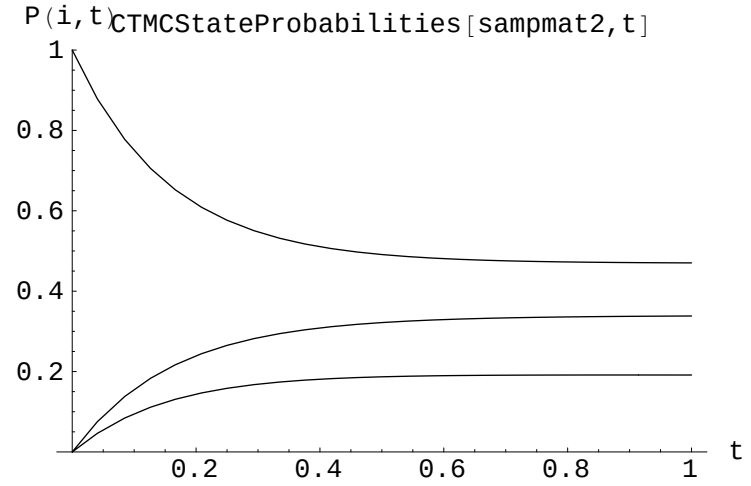
```
(sampmat2  = {{0,     0, 2.1},
              {0,     0, 1.3},
              {2.9, 3.2,   0}}) // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 2.1 \\ 0 & 0 & 1.3 \\ 2.9 & 3.2 & 0 \end{pmatrix}$$

```
sampmat2p=CTMCStateProbabilities[sampmat2,t] //
  Simplify // Chop
```

$$\{0.339909 - 0.3152\, E^{-6.42108\, t} - 0.0247091\, E^{-3.07892\, t},\ 0.190693 - 0.213297\, E^{-6.42108\, t} + 0.0226039\, E^{-3.07892\, t},\ 0.469398 + 0.528497\, E^{-6.42108\, t} + 0.00210525\, E^{-3.07892\, t}\}$$

```
Plot[Evaluate[sampmat2p], {t, 0, 1},
     PlotLabel->"CTMCStateProbabilities[sampmat2,t]",
     AxesLabel->{"t","P(i,t)"},
     PlotRange->{0,1}]
```




```
- Graphics -
```

Note that these solutions contain a transient portion to the solution, which decays to 0 as t approaches Infinity. The asymptotic solution may be abtained with CTMCSteadyStateProbabilities[].

```
(sampmat1b  = {{0, L, 0, 0},
               {0, 0, L, 0},
               {0, 0, 0, L},
               {0, 0, 0, 0}}) // MatrixForm
```

$$\begin{pmatrix} 0 & L & 0 & 0 \\ 0 & 0 & L & 0 \\ 0 & 0 & 0 & L \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
Simplify[CTMCStateProbabilities[sampmat1b,t]] ==
   Simplify[PDPErlangian[3,L,t]]
```

```
True
```

We illustrate with the sampmat1b example above, that CTMCStateProbabilities incorporates the functionality of PDPErlangian as a special case.

```
(sampmat1c  = {{0, 1, 0, 0},
               {0, 0, 2, 0},
               {0, 0, 0, 3},
               {0, 0, 0, 0}}) // MatrixForm
```

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

```
Simplify[CTMCStateProbabilities[sampmat1c,t]] ==
   Simplify[PDPAdjacent[{1,2,3},t]]
```

```
True
```

We illustrate with the sampmat1c example above, that CTMCStateProbabilities incorporates the functionality of PDPAdjacent as a special case.

```
(sampmat1d  = {{0, L[1,0], L[2,0]},
               {0,      0, L[2,1]},
               {0,      0,      0}}) // MatrixForm
```

$$\begin{pmatrix} 0 & L[1, 0] & L[2, 0] \\ 0 & 0 & L[2, 1] \\ 0 & 0 & 0 \end{pmatrix}$$

CTMCStateProbabilities also incorporates the functionality of PDPNonAdjacent as a special case. CTMCStateProbabilities[sampmat1d,t] could be compared to PDPNonAdjacent[2,L,t] (for example) to illustrate this.

■ CTMCMeanAbsorptionTimes[matrix,absorbingstates,initialstate:maximal]

This function will return the mean times to absorption in the given absorbing states, when the system begins in state initialstate. There must be at least one absorbing state either specified in the absorbingstates list, or already present in the transition matrix (identifiable through a zero column). It is assumed that the initial state is not an absorbing state. As this function is discerning the mean time to absorption in ANY absorbing state, it returns one number representing that mean time. As usual, matrix may be symbolic or numerical.

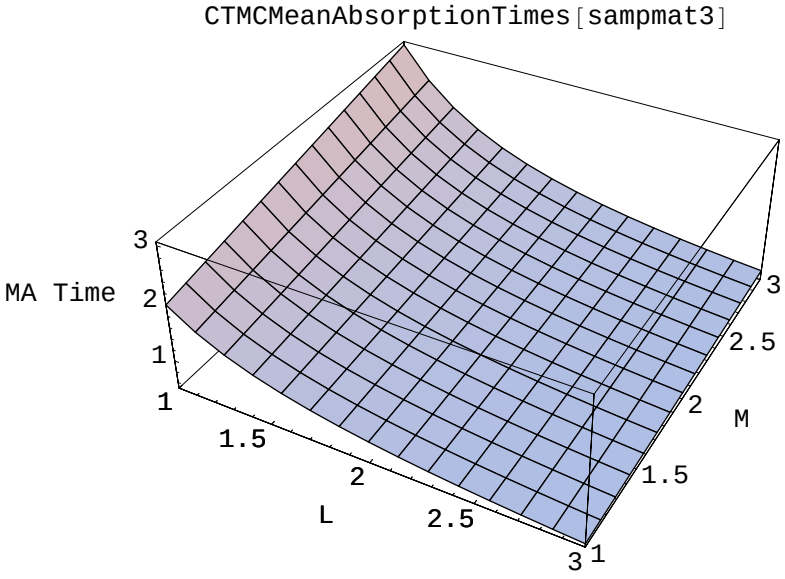
```
(sampmat3  = {{0, L, 0},
              {0, 0, 2L},
              {0, M, 0}}) // MatrixForm
```

$$\begin{pmatrix} 0 & L & 0 \\ 0 & 0 & 2\,L \\ 0 & M & 0 \end{pmatrix}$$

```
sampmat3mat = CTMCMeanAbsorptionTimes[sampmat3] //
   Simplify
```

$$\frac{3\,L + M}{2\,L^2}$$

```
Plot3D[sampmat3mat, {L, 1, 3}, {M, 1, 3},
   PlotLabel->"CTMCMeanAbsorptionTimes[sampmat3]",
   AxesLabel->{"L","M","MA Time"}]
```




```
- SurfaceGraphics -
```

■ CTMCSteadyStateProbabilities[matrix]

Thus function will return the steady-state values for the probabilities of being in each state as a function of time. It returns this inforation in the same form that its non-steady-state equivalent, CTMCStateProbabilities[], does.

```
CTMCSteadyStateProbabilities[sampmat1] // Simplify
```

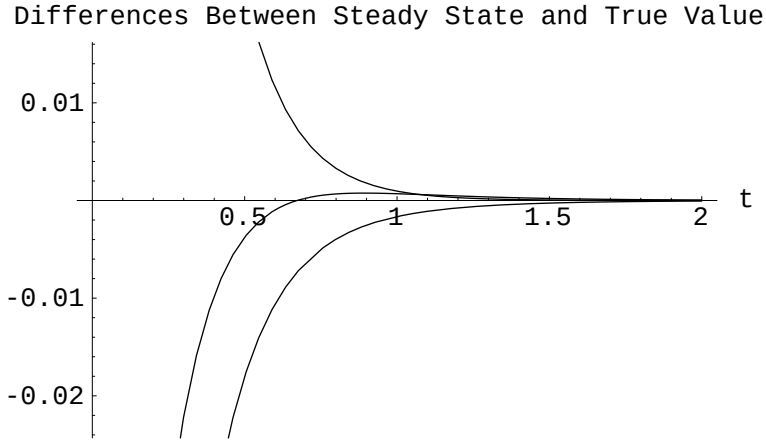
$$\left\{\frac{\text{lambda}}{\text{lambda} + \text{mu}}, \frac{\text{mu}}{\text{lambda} + \text{mu}}\right\}$$

```
sampmat2ps = CTMCSteadyStateProbabilities[sampmat2]
```

```
{0.339909 , 0.190693 , 0.469398 }
```

Please see CTMCStateProbabilitiesfor the definitions of sampmat1 and sampmat2

```
Plot[Evaluate[sampmat2p-sampmat2ps], {t, 0, 2},
 PlotLabel->
  "Differences Between Steady State and True Values",
 AxesLabel->{"t",""}]
```




```
- Graphics -
```

■ CTMCMeanArrivals[matrix]

Thus function will return the expected number of visits ("visits" means arrivals into that state, which is equal to departures from that state) for each state per unit time over a long period of time in steady state. This expression returns meaningful values when there are no absorbing states.

```
(sampmat4  = {{ 0, L1, L2,  0},
              {M1,  0,  0, L2},
              {M2,  0,  0, L1},
              { 0, M2, M1,  0}}) // MatrixForm
```

$$\begin{pmatrix} 0 & L1 & L2 & 0 \\ M1 & 0 & 0 & L2 \\ M2 & 0 & 0 & L1 \\ 0 & M2 & M1 & 0 \end{pmatrix}$$

```
CTMCMeanArrivals[sampmat4] // Simplify
```

$$\left\{\frac{L1\,L2\,(M1+M2)}{(L1+M1)\,(L2+M2)},\ \frac{L2\,M1\,(L1+M2)}{(L1+M1)\,(L2+M2)},\ \frac{L1\,(L2+M1)\,M2}{(L1+M1)\,(L2+M2)},\ \frac{(L1+L2)\,M1\,M2}{(L1+M1)\,(L2+M2)}\right\}$$

■ CTMCStayDurations[matrix]

Thus function will return the expected duration of a stay in each state in steady state. It returns meaningful answers when there are no absorbing states.

```
CTMCStayDurations[sampmat4]
```

$$\left\{\frac{1}{M1+M2},\ \frac{1}{L1+M2},\ \frac{1}{L2+M1},\ \frac{1}{L1+L2}\right\}$$

The Distributions Package

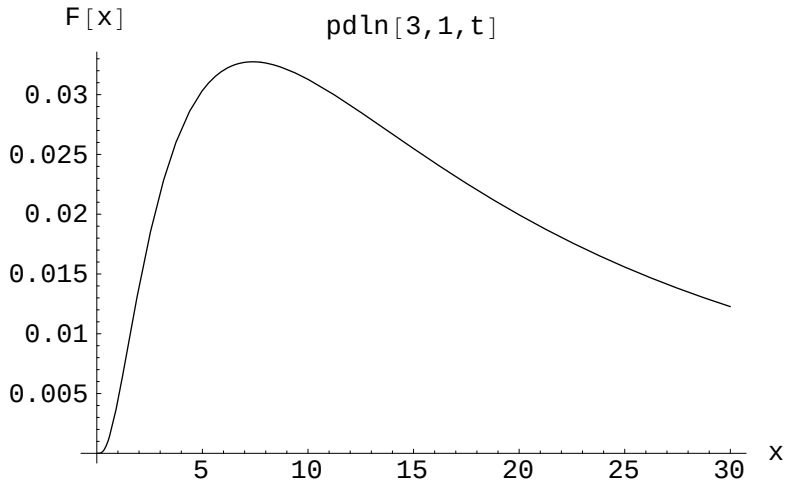
■ General Comments and Definitions

This package is designed to provide access to a variety of distributions (in the form of PDF's, CDF's, and R's, and SYS matrices) that will assist in stochastic modelling for binary, multistate, continuous, and mixed systems. A few additional functions are included which will help set the parameters properly for the Gamma and Weibull distributions, when they are being used as U[t] (lifetime weighting) functions. These last two functions begin with the words "CustomerLimits". Functions which contain "SYS" as a suffix (and FnEstimate) will return SYS matrices. Functions which contain "PDF" as a suffix or which begin with a lower case "p" will return a PDF for a purely continuous distribution (the distinction here is that the "p" PDF's are for random variables with a range [0,Infinity], while the "PDF" PDF's are for random variables with a range [min,max], where min and are both finite). Functions which contain the suffix "CDF" will return general CDF's (functions, rather than matrices). Functions that begin with a lower case "r" are 1-CDF for the same set of distributions

■ Function Documentation

To save space, plots are not shown for these distributions. Any distribution may be plotted by *Mathematica*, with any value for its parameters. The following command, for example, graphs the Log-Normal distribution (pdln) with a mu of 3 and a sigma of 1.

```
Plot[pdln[3,1,t], {t,0,30}, PlotLabel->"pdln[3,1,t]",
   AxesLabel->{"x","F[x]"}]
```




```
- Graphics -
```

■ rdc[n,t]

This function will return R(t) where the distribution is Chi with n degrees of freedom.

```
rdc[n,t]
```

$$1 - \text{GammaRegularized}\left[\frac{n}{2}, 0, \frac{t^2}{2}\right]$$

■ rdcs[n,t]

This function will return R(t) where the distribution is Chi-Square with n degrees of freedom.

```
rdcs[n,t]
```

$$1 - \text{GammaRegularized}\left[\frac{n}{2}, 0, \frac{t}{2}\right]$$

■ rde[lambda,t]

This function will return R(t) where the distribution is Exponential with n degrees of freedom.

```
rde[lambda,t]
```

$$E^{-\text{lambda } t}$$

■ rdfr[n1,n2,t]

This function will return R(t) where the distribution is F with n1 and n2 degrees of freedom.

```
rdfr[n1,n2,t]
```

$$1 - \text{BetaRegularized}\left[\frac{n2}{n2 + n1\ t}, 1, \frac{n2}{2}, \frac{n1}{2}\right]$$

■ rdg[alpha,beta,t]

This function will return R(t) where the distribution is Gamma with parameters alpha and beta.

```
rdg[alpha,beta,t]
```

$$1 - \text{GammaRegularized}\left[\text{alpha}, 0, \frac{t}{\text{beta}}\right]$$

■ rdhn[theta,t]

This function will return R(t) where the distribution is Half-Normal with parameters theta.

```
rdhn[theta,t]
```

$$1 - \text{Erf}\left[\frac{t\ \text{theta}}{\sqrt{\pi}}\right]$$

■ rdln[mu,sigma,t]

This function will return R(t) where the distribution is Log-Normal with parameters mu and sigma.

rdln[mu,sigma,t]

$$1+\frac{1}{2}\left(-1-\text{Erf}\left[\frac{-\text{mu}+\text{Log}[t]}{\sqrt{2}\ \text{sigma}}\right]\right)$$

■ rdncs[n,lambda,t]

This function will return R(t) where the distribution is Non-Central Chi Square with parameters n and lambda.

rdncs[n,lambda,t]

$$1-2^{-n/2}\int_0^t E^{\frac{1}{2}(-\text{lambda}-t1)}\ t1^{-1+\frac{n}{2}}\ \text{Hypergeometric0F1Regularized}\left[\frac{n}{2},\frac{\text{lambda}\ t1}{4}\right]\,dt1$$

■ rdnfr[n1,n2,lambda,t]

This function will return R(t) where the distribution is Non-Central F with parameters n1, n2, and lambda.

rdnfr[n1,n2,lambda,t]

$$1-\left(E^{-\text{lambda}/2}\ n1^{n1/2}\ n2^{n2/2}\ \text{Gamma}\left[\frac{n1}{2}\right]\text{Gamma}\left[1+\frac{n2}{2}\right]\int_0^t E^{\frac{\text{lambda}\ n1\ t2}{2(n2+n1\ t2)}}\ t2^{\frac{1}{2}(-2+n1)}\ (n2+n1\ t2)^{\frac{1}{2}(-n1-n2)}\ \text{LaguerreL}\left[\frac{n2}{2},-1+\frac{n1}{2},-\frac{\text{lambda}\ n1\ t2}{2(n2+n1\ t2)}\right]dt2\right)\Big/\left(\text{Beta}\left[\frac{n1}{2},\frac{n2}{2}\right]\text{Gamma}\left[\frac{n1+n2}{2}\right]\right)$$

■ rdr[sigma,t]

This function will return R(t) where the distribution is Rayleigh with parameter sigma.

```
rdr[sigma,t]
```

$$E^{-\frac{t^2}{2\,sigma^2}}$$

■ rdw[alpha,beta,t]

This function will return R(t) where the distribution is Weibull with parameter alpha and beta.

```
rdw[alpha,beta,t]
```

$$E^{-\left(\frac{t}{beta}\right)^{alpha}}$$

■ rdu[max,t]

This function will return R(t) where the distribution is Uniform, between 0 and max.

```
rdu[max,t]
```

$$1+\frac{1}{2}\left(-1+\frac{(max-t)\,Sign[max-t]}{max}-\frac{t\,Sign[t]}{max}\right)$$

■ pdc[n,t]

This function will return f(t) where the distribution is Chi with n degrees of freedom.

```
pdc[n,t]
```

$$\frac{2^{1-\frac{n}{2}}\,E^{-\frac{t^2}{2}}\,t^{-1+n}}{Gamma\left[\frac{n}{2}\right]}$$

■ pdcs[n,t]

This function will return f(t) where the distribution is Chi-Square with n degrees of freedom.

```
pdcs[n,t]
```

$$\frac{2^{-n/2}\,E^{-t/2}\,t^{-1+\frac{n}{2}}}{Gamma\left[\frac{n}{2}\right]}$$

■ pde[lambda,t]

This function will return f(t) where the distribution is Exponential with n degrees of freedom.

pde[lambda,t]

$$E^{-\text{lambda}\,t}\ \text{lambda}$$

■ pdfr[n1,n2,t]

This function will return f(t) where the distribution is F with n1 and n2 degrees of freedom.

pdfr[n1,n2,t]

$$\frac{n1^{n1/2}\ n2^{n2/2}\ t^{-1+\frac{n1}{2}}\ (n2+n1\,t)^{\frac{1}{2}(-n1-n2)}}{\text{Beta}\left[\frac{n1}{2},\ \frac{n2}{2}\right]}$$

■ pdg[alpha,beta,t]

This function will return f(t) where the distribution is Gamma with parameters alpha and beta.

pdg[alpha,beta,t]

$$\frac{\text{beta}^{-\text{alpha}}\ E^{-\frac{t}{\text{beta}}}\ t^{-1+\text{alpha}}}{\text{Gamma}[\text{alpha}]}$$

■ pdhn[theta,t]

This function will return f(t) where the distribution is Half-Normal with parameters theta.

pdhn[theta,t]

$$\frac{2\,E^{-\frac{t^2\,\text{theta}^2}{\pi}}\ \text{theta}}{\pi}$$

■ pdln[mu,sigma,t]

This function will return f(t) where the distribution is Log-Normal with parameters mu and sigma.

pdln[mu,sigma,t]

$$\frac{E^{-\frac{(-mu+Log[t])^2}{2\,sigma^2}}}{\sqrt{2\,\pi}\;sigma\;t}$$

■ pdncs[n,lambda,t]

This function will return f(t) where the distribution is Non-Central Chi Square with parameters n and lambda.

pdncs[n,lambda,t]

$$2^{-n/2}\,E^{\frac{1}{2}(-lambda-t)}\,t^{-1+\frac{n}{2}}\,\text{Hypergeometric0F1Regularized}\left[\frac{n}{2},\frac{lambda\;t}{4}\right]$$

■ pdnfr[n1,n2,lambda,t]

This function will return f(t) where the distribution is Non-Central F with parameters n1, n2, and lambda.

pdnfr[n1,n2,lambda,t]

$$\frac{1}{\text{Beta}\left[\frac{n1}{2},\frac{n2}{2}\right]}\left(E^{-lambda/2}\,n1^{n1/2}\,n2^{n2/2}\,t^{\frac{1}{2}(-2+n1)}\,(n2+n1\,t)^{\frac{1}{2}(-n1-n2)}\;\text{Hypergeometric1F1}\left[\frac{n1+n2}{2},\frac{n1}{2},\frac{lambda\;n1\;t}{2\,(n2+n1\,t)}\right]\right)$$

■ pdr[sigma,t]

This function will return f(t) where the distribution is Rayleigh with parameter sigma.

pdr[sigma,t]

$$\frac{E^{-\frac{t^2}{2\,sigma^2}}\;t}{sigma^2}$$

■ pdw[alpha,beta,t]

This function will return f(t) where the distribution is Weibull with parameter alpha and beta.

pdw[alpha,beta,t]

$$\text{alpha beta}^{-\text{alpha}}\, E^{-\left(\frac{t}{\text{beta}}\right)^{\text{alpha}}}\, t^{-1+\text{alpha}}$$

■ pdu[max,t]

This function will return f(t) where the distribution is Uniform, between 0 and max.

pdu[max,t]

$$\frac{\text{Sign}[t] - \text{Sign}[-\text{max} + t]}{2\,\text{max}}$$

■ BinaryCDF[x,p]

This function will return the probability of a component or system being in or below the given state x, given that its allowable states are {0,1}, and that the probability of the unit being in state 1 is p. For this function, the techniques of making the parameters functions of time, plotting, etc. will be demonstrated. The techniques will work in the same way for all functions in that package with a suffix "CDF", and may not be demonstrated anew for each subsequent function.

BinaryCDF[1/2,3/4]

$$\frac{1}{4}$$

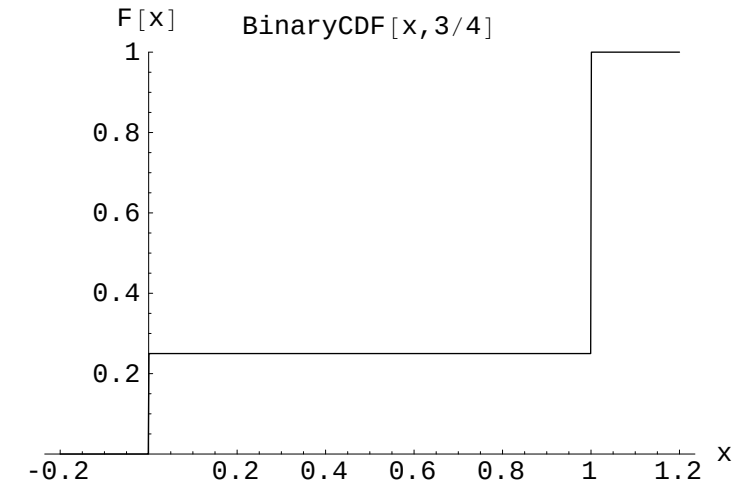
BinaryCDF[-1,3/4]

0

BinaryCDF[1,3/4]

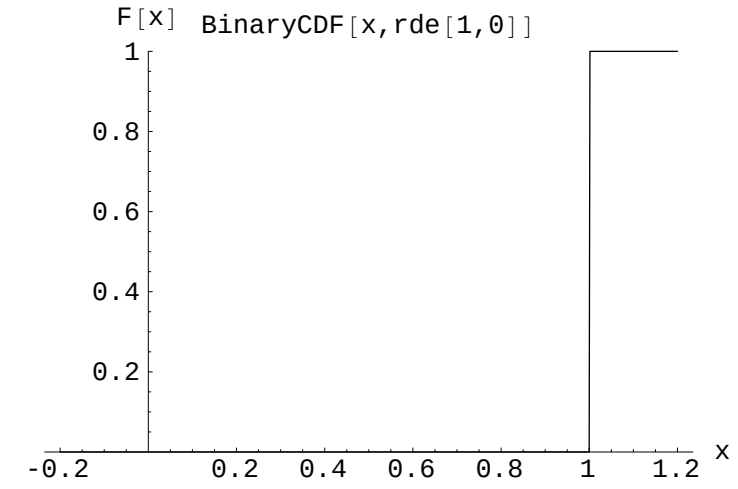
1

```
Plot[BinaryCDF[x,3/4],{x,-0.2,1.2},
     PlotLabel->"BinaryCDF[x,3/4]",
     AxesLabel->{"x","F[x]"},
     PlotRange->{0,1}]
```



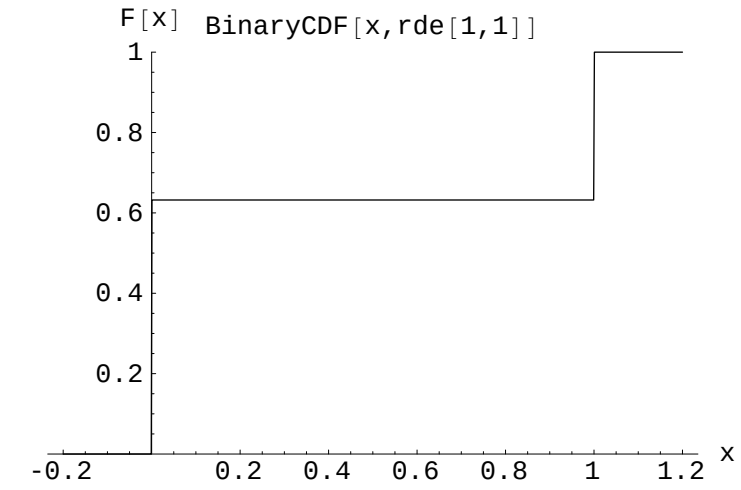
- Graphics -

```
bincdf=BinaryCDF[x,rde[1,t]];

Plot[bincdf /. t->0,{x,-0.2,1.2},
   PlotLabel->"BinaryCDF[x,rde[1,0]]",
   AxesLabel->{"x","F[x]"},
   PlotRange->{0,1}]
```



- Graphics -

```
Plot[bincdf /. t->1,{x,-0.2,1.2},
   PlotLabel->"BinaryCDF[x,rde[1,1]]",
   AxesLabel->{"x","F[x]"},
   PlotRange->{0,1}]
```




```
- Graphics -
```

■ MultistateCDF[x,sys]

This function will return the probability of a component or system being in or below the given state x, given that its allowable states are as given by sys (a regular multistate system matrix). Depending on how sys is defined, the domain of the random variable in question is not necessarily [0,1].

```
multsys=SystemMatrix[Range[0,3]/3,PDPErlangian[3,2,t]]
```

$$\left\{\{0,\ 1-E^{-2t}-2E^{-2t}t-2E^{-2t}t^2\},\ \left\{\frac{1}{3},\ 2E^{-2t}t^2\right\},\ \left\{\frac{2}{3},\ 2E^{-2t}t\right\},\ \{1,\ E^{-2t}\}\right\}$$

Note that in the sys matrix defined above, the state values are not necessarily integers. In general, in RelPack, one is allowed to and encouraged to specify state values for components and systems that are reflective of the customer satisfaction associated with each level, rather than state values which reflect only the ordinal number of that state. When one specifies a system state as a parameter input to a function, in general one specifies the actual state value rather than its ordinal number. The approach (philosophically) with RelPack is that the binary system states are not 0,1 because 1 is the first integer above 0, but rather because 0 represents 0% customer satisfaction and 1 represents 100% customer satisfaction. Thus, for most RelPack work, the state values may (and should) be non-integers, with a maximal value of 1 and a minimal value of 0 (these maximal and minimal values are common defaults for many functions).

```
MultistateCDF[1/2,multsys]
```

$$1 - E^{-2t} - 2E^{-2t}t$$

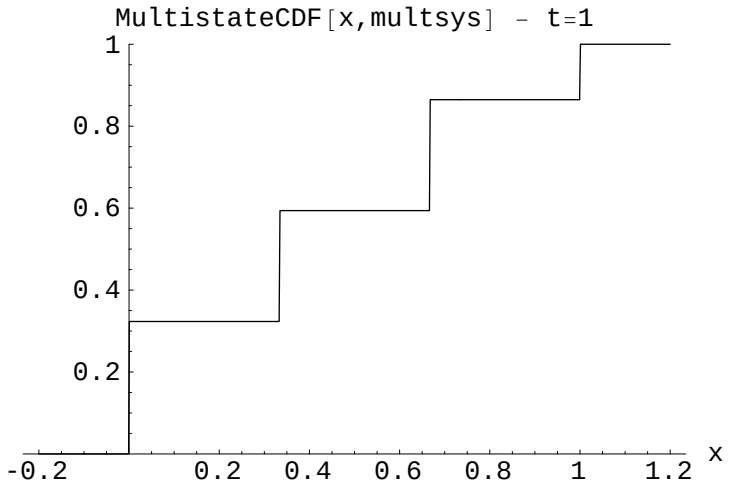
```
multsys[[1,2]]+multsys[[2,2]] ==
   MultistateCDF[1/2,multsys]
```

```
True
```

```
multsys
```

$$\left\{\{0,\ 1 - E^{-2t} - 2E^{-2t}t - 2E^{-2t}t^2\},\ \left\{\frac{1}{3},\ 2E^{-2t}t^2\right\},\ \left\{\frac{2}{3},\ 2E^{-2t}t\right\},\ \{1,\ E^{-2t}\}\right\}$$

```
Plot[MultistateCDF[x,multsys] /. t->1,{x,-0.2,1.2},
   PlotLabel->"MultistateCDF[x,multsys] - t=1",
   AxesLabel->{"x",""},
   PlotRange->{0,1}]
```



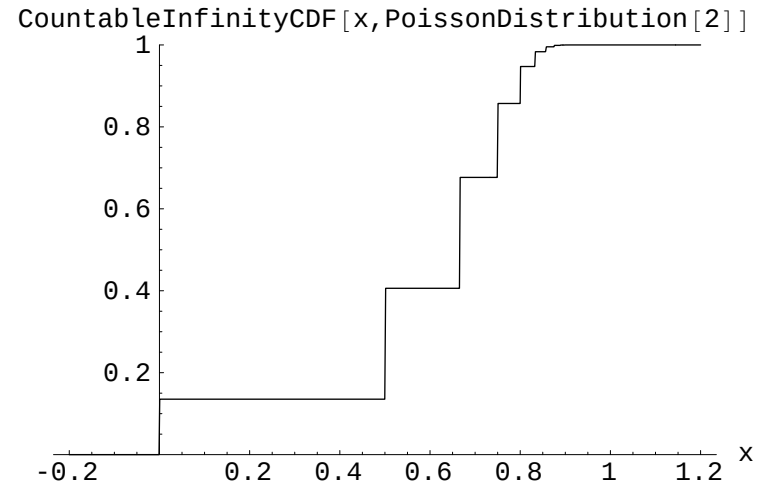

```
- Graphics -
```

■ CountableInfinityCDF[x,mmadist]

This function will accept either GeometricDistribution[p] or NegativeBinomialDistribution[n,p] or PoisonDistribution[mu] and map the states of discrete distributions into the range [0,1], returning a CDF reflecting this new distribution. Note that one's choices for "mmadist" in this case are limited to those in the Statistics`DiscreteDistributions` standard Mathematica package.

```
cicdf = CountableInfinityCDF[x,PoissonDistribution[2]];
```

```
Plot[cicdf,{x,-0.2,1.2},
   PlotLabel->
      "CountableInfinityCDF[x,PoissonDistribution[2]]",
   AxesLabel->{"x",""},
   PlotRange->{0,1}]
```




```
- Graphics -
```

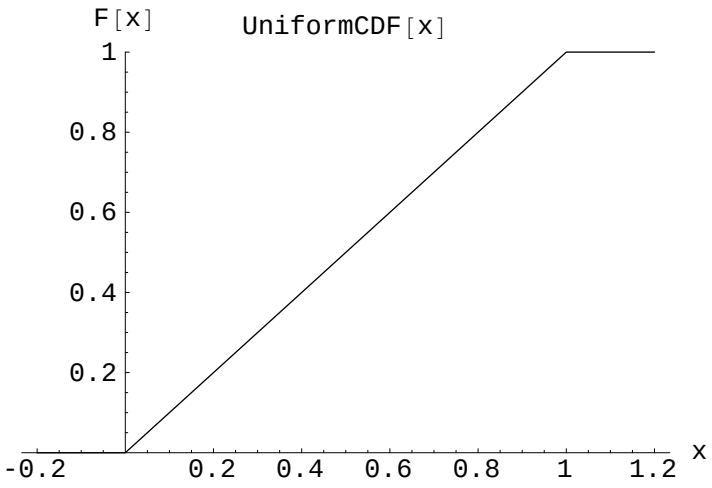
■ UniformCDF[x]

This function will return the probability of a uniformly distributed continuous random variable being less than or equal to x, where the domain of the random variable is [0,1].

```
UniformCDF[1/2]
```

$$\frac{1}{2}$$

```
Plot[UniformCDF[x],{x,-0.2,1.2},
   PlotLabel->"UniformCDF[x]",
   AxesLabel->{"x","F[x]"},
   PlotRange->{0,1}]
```

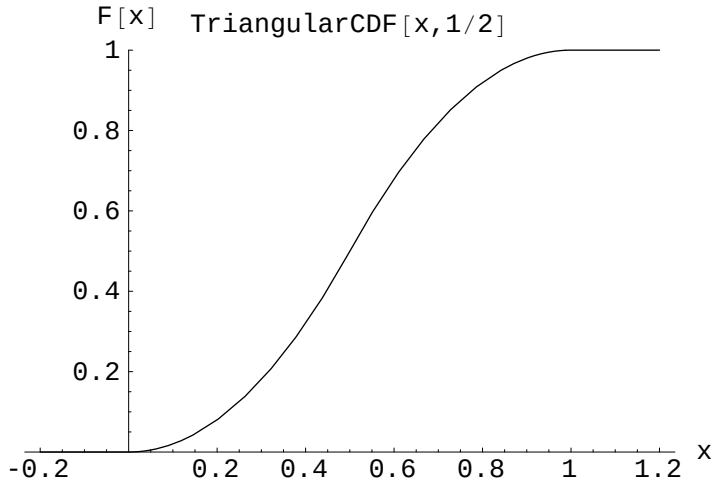


- Graphics -

■ TriangularCDF[x,a:1/2]

This function will return the probability that a unit is in or below the given state x, given that the distribution in question is triangular on the domain [0,1], and that the mode of that distribution is a.

```
Plot[TriangularCDF[x,1/2],{x,-0.2,1.2},
   PlotLabel->"TriangularCDF[x,1/2]",
   AxesLabel->{"x","F[x]"},
   PlotRange->{0,1}]
```

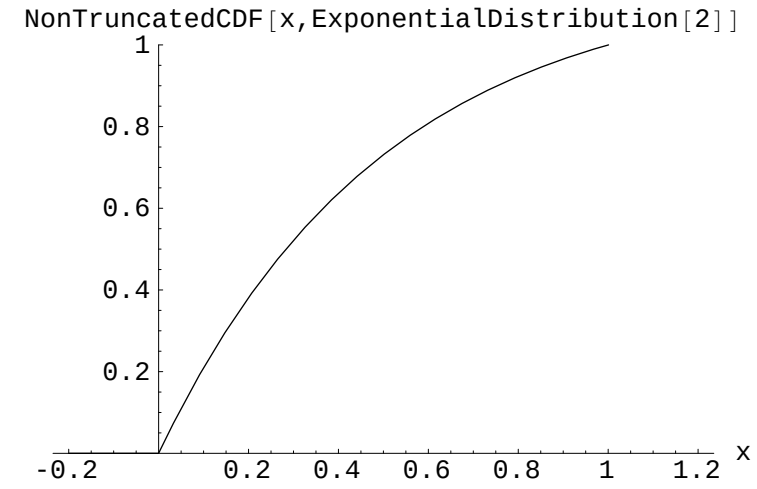



```
- Graphics -
```

■ NonTruncatedCDF[x,mmadist]

This function will return the probability of being in or below the given state x, for a distribution that has been scaled to lie in [0,1]. This is distinguised from TruncatedCDF, in that TruncatedCDF takes the probability of being in or above state 1 and adds that to the probability of being in state 1, and takes the probability of being in or below state 0 and adds that to the probability of being in state 0. NonTruncatedCDF linearly scales the CDF so that the probabilities of being outside of [0,1] are even applied throughout [0,1]. Note that mmadist may be any Mathematica distribution, discrete or continuous. However, if a discrete distribution is used, it should have at least some possible values in the range [0,1].

```
Plot[NonTruncatedCDF[x,ExponentialDistribution[2]],
 {x,-0.2,1.2},
 PlotLabel->"NonTruncatedCDF[x,ExponentialDistribution[2]]",
 AxesLabel->{"x",""},
 PlotRange->{0,1}]
```

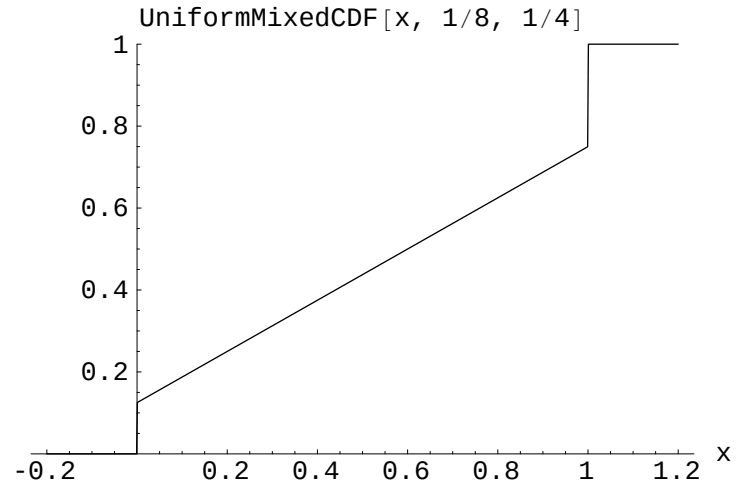



```
- Graphics -
```

■ UniformMixedCDF[x,li:0, ui:0]

This function returns the probability of a system being in or below the given state x, given that its allowable states are [0,1], the probability of the system being in state 1 is ui, the probability of the system being in state 0 is li, and the probabilities of being in any state in (0,1) are governed by a uniform continuous distribution.

```
Plot[UniformMixedCDF[x, 1/8, 1/4], {x,-0.2,1.2},
   PlotLabel->"UniformMixedCDF[x, 1/8, 1/4]",
   AxesLabel->{"x",""},
   PlotRange->{0,1}]
```

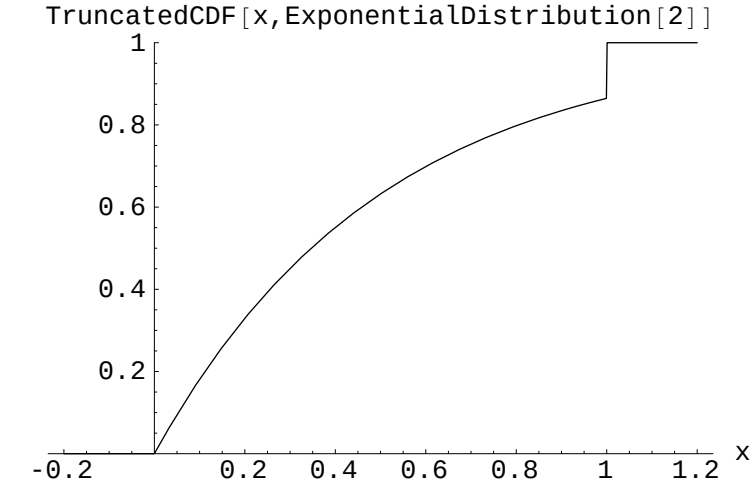



```
- Graphics -
```

■ TruncatedCDF[x,mmadist]

See NonTruncatedCDFfor a discussion of this TruncatedCDF,and for a discussion of the differences between TruncatedCDFand NonTruncatedCDF.

```
Plot[TruncatedCDF[x,ExponentialDistribution[2]],
   {x,-0.2,1.2},
   PlotLabel->
    "TruncatedCDF[x,ExponentialDistribution[2]]",
   AxesLabel->{"x",""},
   PlotRange->{0,1}]
```

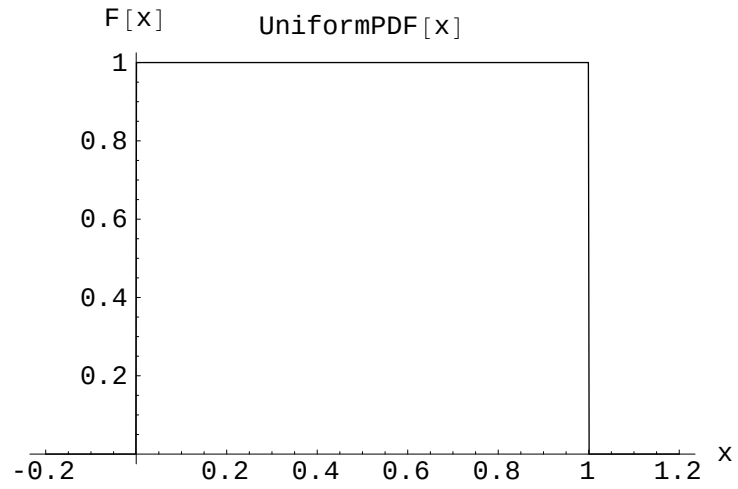



```
- Graphics -
```

■ UniformPDF[x,min:0, max:1]

This function will return the PDF for a uniform continuous random variable which has the domain [min, max].

```
Plot[UniformPDF[x], {x,-0.2,1.2},
   PlotLabel->"UniformPDF[x]",
   AxesLabel->{"x","F[x]"}]
```

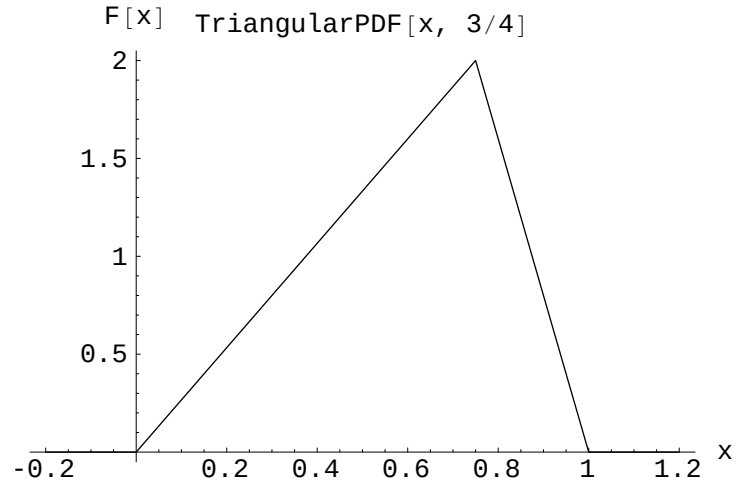



```
- Graphics -
```

■ TriangularPDF[x,a:1/2, min:0, max:1]

This function will return the PDF for a triangular random variable with mode a, and domain [min,max].

```
Plot[TriangularPDF[x, 3/4], {x,-0.2,1.2},
   PlotLabel->"TriangularPDF[x, 3/4]",
   AxesLabel->{"x","F[x]"}]
```




```
- Graphics -
```

■ BernoulliSYS[p]

This function will return the SYS multistate system for a Bernoulli random variable which is in state 1 with probability p and state 0 with probability (1-p). Essentially, this is the same random variable as BinaryCDF, but is returned in SYS form rather than CDF form.

```
BernoulliSYS[rde[2,t]]
```

```
{{0, 1 - E^-2t}, {1, E^-2t}}
```

■ UniformDiscreteSYS[n]

This function will return the multistate SYS system for a discrete random variable which has n discrete outcomes, each of which is equally likely, and where the events are evenly spaced in [0,1].

```
UniformDiscreteSYS[5]
```

$$\left\{\left\{0, \frac{1}{5}\right\}, \left\{\frac{1}{4}, \frac{1}{5}\right\}, \left\{\frac{1}{2}, \frac{1}{5}\right\}, \left\{\frac{3}{4}, \frac{1}{5}\right\}, \left\{1, \frac{1}{5}\right\}\right\}$$

■ BinomialSYS[n,p]

This function will return the multistate SYS system for a discrete random variable with the Binomial distribution, where the states have been mapped into [0,1].

```
BinomialSYS[5,1/3]
```

$$\left\{\left\{0, \frac{32}{243}\right\}, \left\{\frac{1}{5}, \frac{80}{243}\right\}, \left\{\frac{2}{5}, \frac{80}{243}\right\}, \left\{\frac{3}{5}, \frac{40}{243}\right\}, \left\{\frac{4}{5}, \frac{10}{243}\right\}, \left\{1, \frac{1}{243}\right\}\right\}$$

■ HypergeometricSYS[n,nsucc,notot]

This function will return the multistate SYS system for a discrete random variable with the Hypergeometricdistribution (parametersn, nsucc, notot), where the states have been mapped into [0,1].

```
HypergeometricSYS[2,3,5]
```

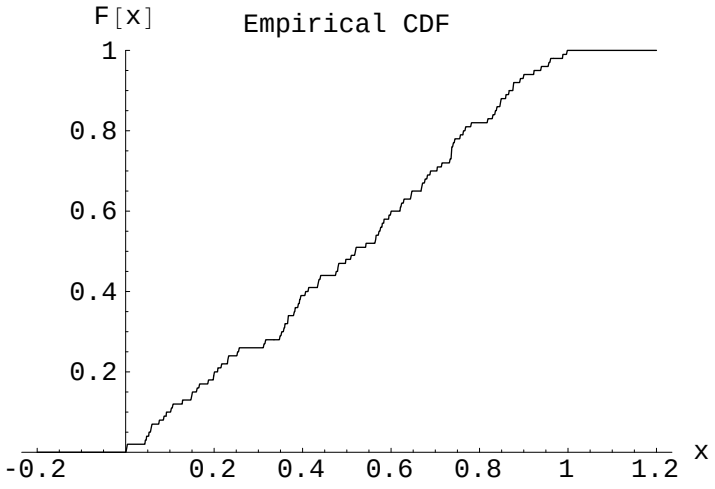
$$\left\{\left\{0, \frac{1}{10}\right\}, \left\{\frac{1}{2}, \frac{3}{5}\right\}, \left\{1, \frac{3}{10}\right\}\right\}$$

■ FnEstimate[list]

This function will return the SYS multistate system which is empirically found based on the list of data contained in list. The resulting sys system can then be used as a sys matrix, or passed to MultistateCDF[x,sys] to form a true CDF.

```
list = Table[Random[], {100}];

empcdf = MultistateCDF[x,FnEstimate[list]];
```

```
Plot[empcdf // Evaluate,{x,-0.2,1.2},
   PlotLabel->"Empirical CDF",
   AxesLabel->{"x","F[x]"}, PlotRange->{0,1}]
```




```
- Graphics -
```

The following is an example of how one would create a multistate system which is based on time based data taken over time. The data is in the form of the state of each of a series of items on test in one list, with one list every time interval dt for the duration of the test.

```
TestSYS[t_ /; NumberQ[t]] := With[{
      test={{1, 1, 1}, {0.8, 0.9, 0.7},
            {0.6, 0.6, 0.5}, {0.3, 0.4, 0.2}},
      dt=1/2},
    FnEstimate[test[[Floor[t/dt]+1]]]]

sys5 = TestSYS[t];
```

One would invoke this construction in a manner as follows:

```
sys5 /. t->1
```

$$\left\{\left\{0.5,\ \frac{1}{3}\right\},\ \left\{0.6,\ \frac{2}{3}\right\}\right\}$$

■ FnEstimateVar[list]

This function will return the variance of the FnEstimate estimate of a random variable X, based on the input of n sample data poits to FnEstimate. It is an upper bound, independent of x.

```
FnEstimateVar[list]
```

$$\frac{1}{400}$$

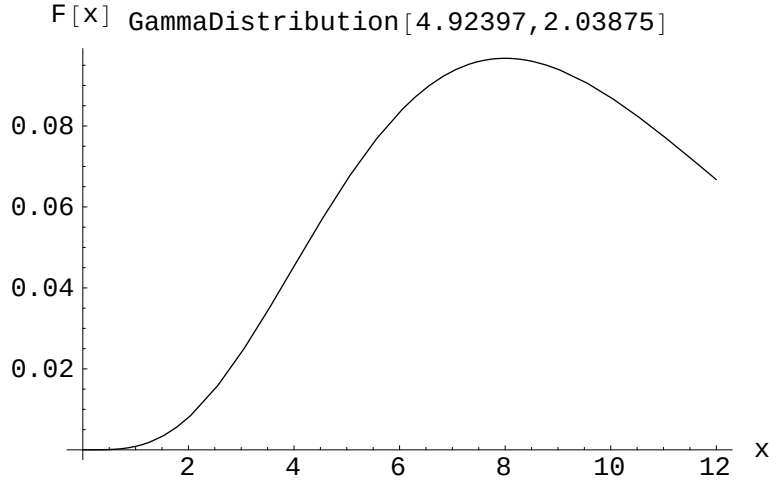
■ CustomerLimitsGamma[m,x:1/2, a:0, b:m]

This function will return the pair of arguments which should be passed to GammaDistribution, if one wishes to have a distribution with mode m and the fraction x of the total area underneath the PDF curve to lie between b and a. The motivation here is that, in creating U[t] lifetime weighting functions, it would be more natural for the customer to specify the mode than the mean for the function (which is assumed to take the form of some PDF). For most PDFs, knowing the mode uniquely specifies the necessary parameters for the distribution. However, this is not the case for the Gamma and Weibull distributions (for example). To pin down all the parameters in these cases, these functions will allow the customer to specify the mode as well as CDF value at some point. An optimization routine then attempts to discover the appropriate parameters. Note that it is possible to specify problems for which there is no feasible solution.

```
CustomerLimitsGamma[8, 0.37]
```

```
{4.92397, 2.03875}
```

```
Plot[PDF[GammaDistribution[4.92397,2.03875],x],
     {x,0,12},
   PlotLabel->"GammaDistribution[4.92397,2.03875]",
   AxesLabel->{"x","F[x]"}]
```




```
- Graphics -
```

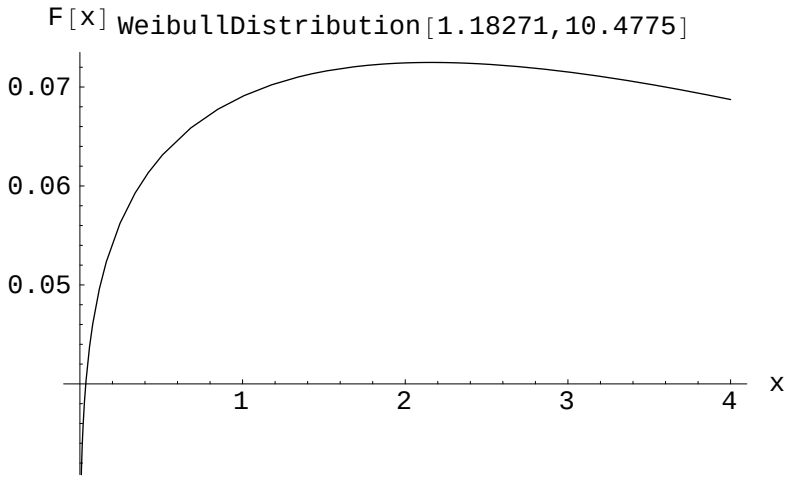
■ CustomerLimitsWeibull[m,x:1/2, a:0, b:m]

Please see CustomerLimitsGammafor further discussion of this function. This function considers the Weibull distribution rather than the Gamma, but in all other respects is the same. Please keep in mind again that it is possible to specific infeasible parameters to this function.

```
CustomerLimitsWeibull[2.16, 0.027, 0, 1/2]
```

```
{1.18271, 10.4775}
```

```
Plot[PDF[WeibullDistribution[1.18271,10.4775],x],
        {x,0,4},
   PlotLabel->"WeibullDistribution[1.18271,10.4775]",
   AxesLabel->{"x","F[x]"}]
```

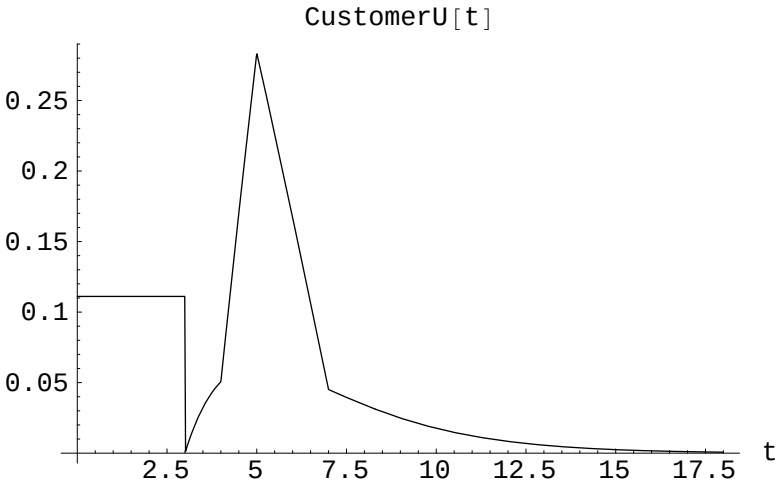



```
- Graphics -
```

Along the lines of discussing how to form different types of U[t] functions, the following is an illustration of how one would create a U[t] function which is a mixture of a variety of CDF's and displacements of CDF's. Note that the last PDF in the set of three that define CustomerU is shifted to the right by three time units. Building functions in this way is a good way to account for cyclic and projected demands.

```
CustomerU[t_] := (TriangularPDF[t,5,4,7] +
                  PDF[UniformDistribution[0,3],t] +
                  PDF[GammaDistribution[2,2],t-3])/3
```

```
Plot[CustomerU[t],{t,0,18}, PlotLabel->"CustomerU[t]",
   AxesLabel->{"t",""}]
```



- Graphics -

```
NIntegrate[CustomerU[t], {t, 0, Infinity},
   AccuracyGoal->3]
```

```
1.00006
```

The Measures Package

■ General Comments and Definitions

This package contains functions which will calculate various reliability measures based on information about the stochastic behavior of the system or component under consideration. Some functions accept this input in the form of "SYS" matrices (for discrete systems), and some functions accept this input in the form of "CDF" functions (for continuous and mixed systems, and for components of discrete systems that are specified by CDF only so as to retain full generality should other components be non-discrete). As it is clear from the information in the parameter list for each function whether it is exclusively discrete or not, this will not generally be commented on in the documentation for each function.

As usual, SYS is in the form {{phi_0, P[phi=phi_0]}, {phi_1, P[phi=phi_1]}, ..., {phi_M, P[phi=phi_M]}. As usual, it is assumed that none of of the phi_i are the same, and that they are given (with their associated probabilities) in ascending order (i.e. they are listed from worst to best from the point of the customer). As usual, both SYS matrices and CDF'smay be functions of time, and there are many dynamic measures in this package that integrate or similarly consider the changing behavior over time.

There are only two functions in this package which do not take either SYS or CDF as input. The first is ContinuousEntropy, which takes takes a PDF and is valid only for completely continuous distributions. The second is ChebyshevUB, does not use distributions of any kind (only the variance information).

Many of the discrete functions involving integration allow the user to pass the function in question a final parameter ni, which must be either True or False. If the parameter is False or omitted, then the regular symbolic calculation is attempted. If only a numerical result is desired, then much computation time can be saved by setting this value equal to True.

One point should be made quite strongly. When one may specify the distribution of a component or system through its CDF in this package, one may pass a CDF which is mixed (i.e. has both continuous and discrete behavior, and appears as a continuous CDF with periodic "jumps" at the points of dicrete behavior). This was deemed important to the study of continuous systems, as by simple reason a system which is non-repairable should have a growing, non-zero probability of being in its minimal state.

Please note that these measures will operate on any list of states/probabilities (for SYS input) or CDF (for CDF input). Whether those probabilities and states are states of a system, or states or

subsystems, or states of components, is mostly immaterial. Whenever one reads "system" in this document, one could think equivalently of "subsystem" or "component", as the case may be.

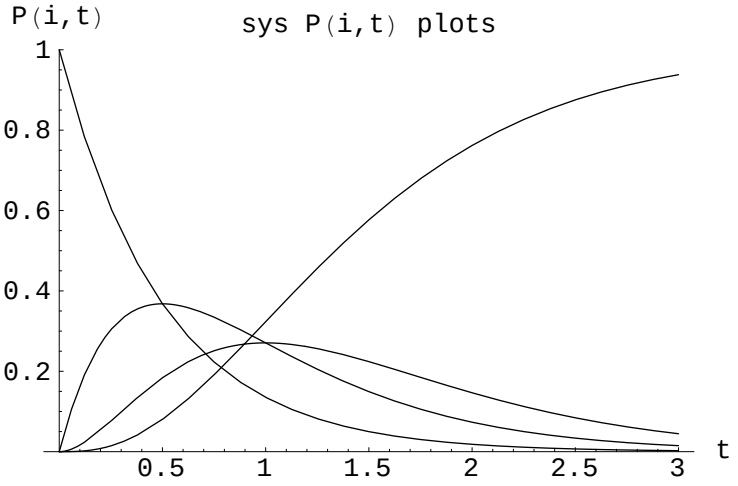
Throughout this document, we will make use of the same set of simple examples upon which to demonstrate the use of the different reliability measures. The SYS example is one where there are four distinct states to the system, ranging from 0 (worst) to 1 (best). This non-repairable system begins at t=0 in its best state, and spends a length of time in each state>0 governed by an exponential distribution, with parameter lambda=2, before moving to the next lower state.

```
sys=SystemMatrix[Range[0,3]/3,PDPErlangian[3,2,t]] //
    Simplify
```

$$\left\{\{0,\ E^{-2\,t}\ (-1+E^{2\,t}-2\,t-2\,t^2)\},\ \left\{\frac{1}{3},\ 2\,E^{-2\,t}\,t^2\right\},\ \left\{\frac{2}{3},\ 2\,E^{-2\,t}\,t\right\},\ \{1,\ E^{-2\,t}\}\right\}$$

Here is a plot of the probabilities of this system being in any one of its states, as a function of time.

```
Plot[Evaluate[PDPErlangian[3,2,t]], {t, 0, 3},
   PlotLabel->"sys P(i,t) plots",
   AxesLabel->{"t","P(i,t)"},
   PlotRange->{0,1}]
```

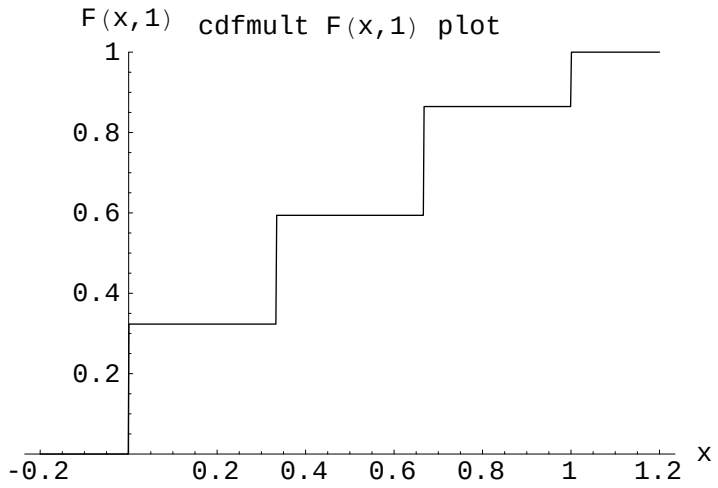



```
- Graphics -
```

A series of continuous and mixed distributions will be defined, so as to have one at any given time that is appropriate for the example at hand. The first is the cdf version of sys, and will be used to demonstrate that the cdf functions work equivalently with discrete systems.

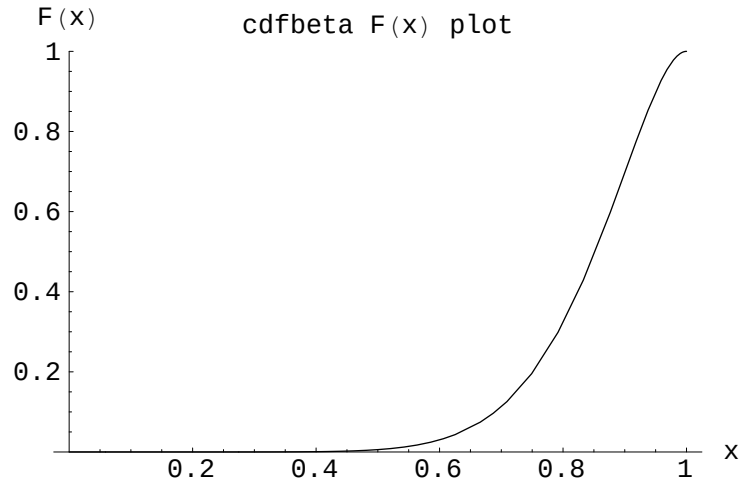
```
cdfmult = MultistateCDF[x,SystemMatrix[Range[0,3]/3,
   PDPErlangian[3,2,t]]];
```

```
Plot[cdfmult /. t->1 // Evaluate, {x, -.2, 1.2},
   PlotLabel->"cdfmult F(x,1) plot",
   AxesLabel->{"x","F(x,1)"},
   PlotRange->{0,1}]
```



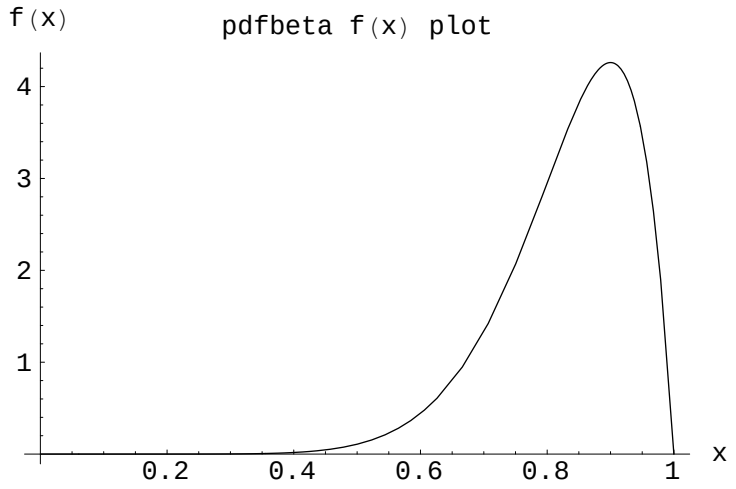
- Graphics -

```
cdfbeta = CDF[BetaDistribution[10,2],x];

Plot[cdfbeta, {x, 0, 1},
   PlotLabel->"cdfbeta F(x) plot",
   AxesLabel->{"x","F(x)"},
   PlotRange->{0,1}]
```



- Graphics -

```
Plot[PDF[BetaDistribution[10,2],x], {x, 0, 1},
   PlotLabel->"pdfbeta f(x) plot",
   AxesLabel->{"x","f(x)"}]
```




```
- Graphics -
```

Note that cdfbeta is a purely continuous distribution, that naturally has the range [0,1]. It has the following PDF:

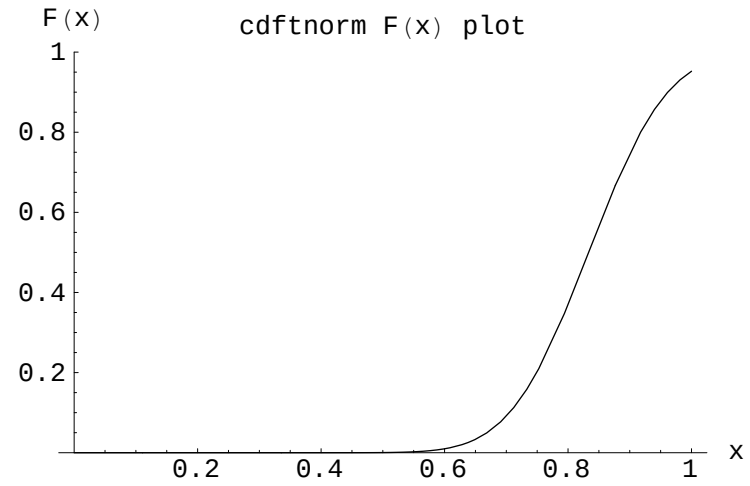
```
pdfbeta = PDF[BetaDistribution[10,2],x]
```

$$110\,(1-x)\,x^9$$

Now, we wish to define a mixed distribution. Let us consider a normal distribution that is truncated to lie within the range [0,1].

```
cdftnorm=TruncatedCDF[x,NormalDistribution[5/6,1/10]];
```

```
Plot[cdftnorm, {x, 0, 1},
   PlotLabel->"cdftnorm F(x) plot",
   AxesLabel->{"x","F(x)"},
   PlotRange->{0,1}]
```

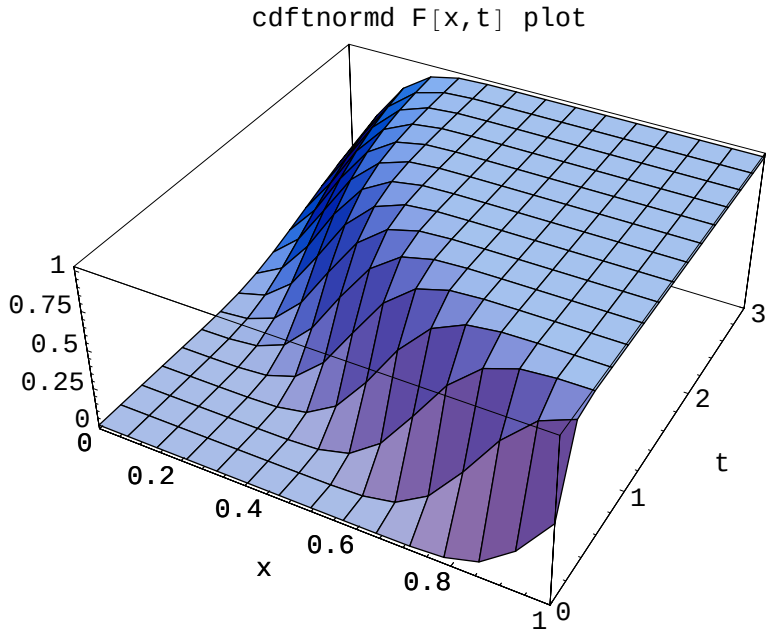



```
- Graphics -
```

Now, let's define a mixed, dynamic distribution.

```
cdftnormd =
   TruncatedCDF[x,NormalDistribution[E^(-t),1/10]];
```

```
Plot3D[cdftnormd, {x,0,1}, {t,0,3},
   PlotLabel->"cdftnormd F[x,t] plot",
   AxesLabel->{"x","t",""}]
```




```
- SurfaceGraphics -
```

■ Function Documentation

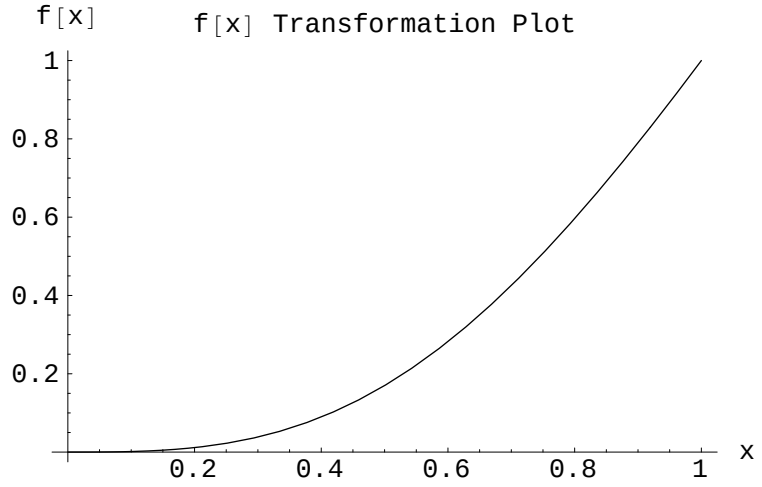
■ ExpectedFnState[sys,f]

This function will return the expected value of a function f of the state of the sys system.

First, let'sdefine the transformationfunction f.

```
f[x_] := Log[x^3+1]/Log[2]
```

```
Plot[f[x], {x, 0, 1},
   PlotLabel->"f[x] Transformation Plot",
   AxesLabel->{"x","f[x]"}]
```

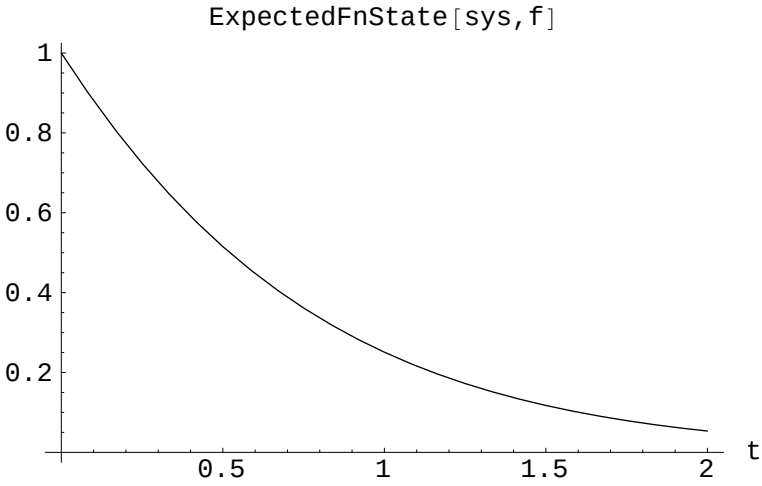


- Graphics -

```
ans1=ExpectedFnState[sys,f] // Simplify
```

$$\frac{E^{-2t}\left(2t^2 \text{Log}\left[\frac{28}{27}\right] + 2t \text{Log}\left[\frac{35}{27}\right] + \text{Log}[2]\right)}{\text{Log}[2]}$$

```
Plot[ans1, {t, 0, 2},
   PlotLabel->"ExpectedFnState[sys,f]",
   AxesLabel->{"t",""}]
```



- Graphics -

```
ans1 /. t->1 // N
```

```
0.250875
```

This may be compared to the simple expected value of the state of the system (see ExpectedState[])

To calculate the equivalent value for the mixed case, use the function StieltjesIntegral[f,cdf, {x, min, max}]. To calculate the equivalent value for the absolutely continuous case when the PDF g[x] is available, use Integrate[f[x]g[x], {x, -Infinity, Infinity}], or NIntegrate for a numerical approximation.

```
StieltjesIntegral[f[x],cdfmult /. t->1,{x},3]
```

```
0.251092
```

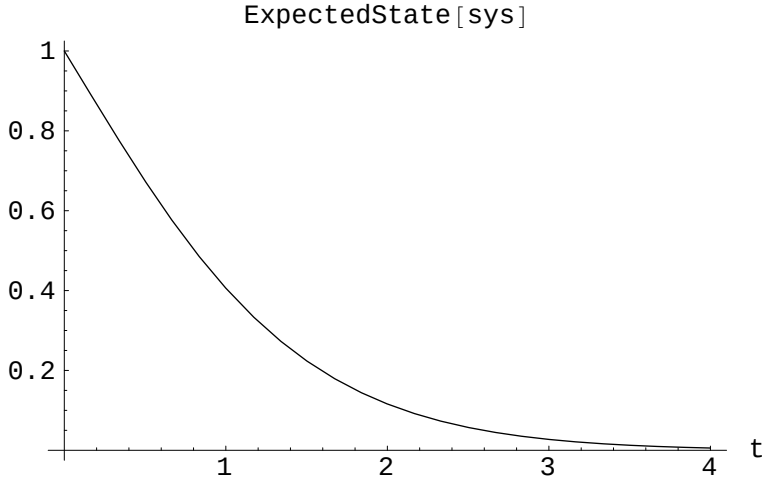
■ ExpectedState[sys]

This function will return the expected state of the system. It has the range [min,max].

```
ans2=ExpectedState[sys] // Simplify
```

$$\frac{1}{3} E^{-2t} (3 + 4t + 2t^2)$$

```
Plot[ans2, {t, 0, 4},
   PlotLabel->"ExpectedState[sys]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

```
ans2 /. t->1 // N
```

```
0.406006
```

■ CDFExpectedState[cdf,x, max:1]

This function will return the expected state of the system. It has the range [min,max]. It is assumed that system state values are non-negative.

```
CDFExpectedState[cdfmult /. t->1, x]
```

```
0.406006
```

```
CDFExpectedState[cdfbeta, x]
```

```
0.833333
```

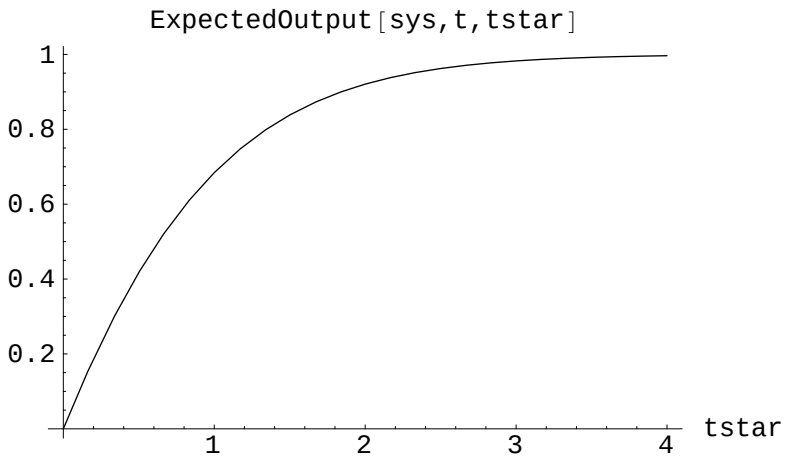
■ ExpectedOutput[sys,t,tstar,ni:False]

This function will return the integral of the expected state of the system over the product's useful lifetime tstar. ni is the flag for Numerical Integration (vs. Symbolic Integration, as mentioned at the beginning of this document). This measure has the range [min*tstar, max*tstar].

```
ans3=ExpectedOutput[sys,t,tstar]
```

$$1 - \frac{1}{3} E^{-2\,\text{tstar}} \left(3 + 3\,\text{tstar} + \text{tstar}^2\right)$$

```
Plot[ans3, {tstar, 0, 4},
   PlotLabel->"ExpectedOutput[sys,t,tstar]",
   AxesLabel->{"tstar",""}]
```




```
- Graphics -
```

```
ans3 /. tstar->1 // N
```

```
0.684218
```

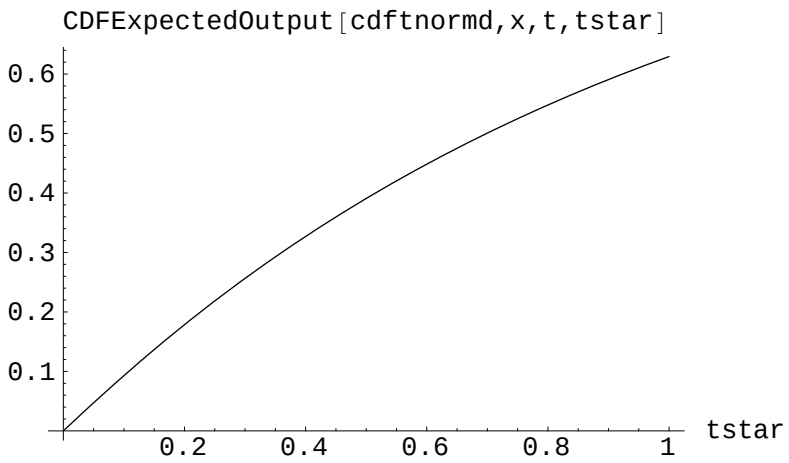
■ CDFExpectedOutput[cdf,x, t, tstar, max:1]

This is the equivalent of ExpectedOutput using the cdf. Calculation of this measure can take significant amounts of time (especially for multistate CDFs), so if the component or system in question is indeed discrete, converting it to a SYS matrix and using ExpectedOutput would be preferable (though not TECHNICALLY necessary).

```
CDFExpectedOutput[cdftnormd,x,t,1]
```

```
0.629473
```

```
Plot[CDFExpectedOutput[cdftnormd,x,t,tstar],
    {tstar, 0, 1},
  PlotLabel->"CDFExpectedOutput[cdftnormd,x,t,tstar]",
  AxesLabel->{"tstar",""}]
```




```
- Graphics -
```

■ ExpectedTotalOutput[sys,t,ni:False]

This function returns the integral of the expected state of the system over all time. It is the value which ExpectedOutputapproaches. Note that this measure will diverge for systems which are reparairableand have a non-zero steady-state, or for systems which have a non-zero minimum state.

```
ExpectedTotalOutput[sys,t]
```

```
1
```

■ CDFExpectedTotalOutput[cdf,x,t,max:1]

This function is the equivalent of ExpectedTotalOutput,but uses the CDF. Calculation of this measure can require considerable amounts of time.

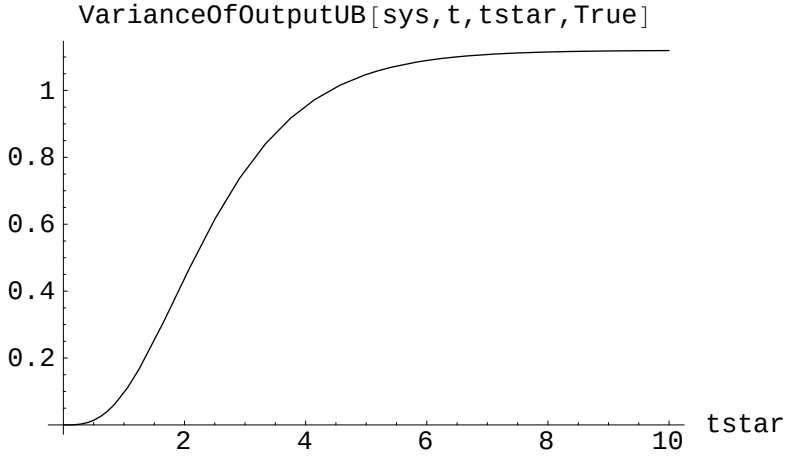
■ VarianceOfOutputUB[sys,t,tstar,ni:False]

This function will find the expression for the upper bound (using the Schwartz inequality) of the variance of the integral of the state of the system from time 0 to time tstar. ExpectedOutputwas the mean, rather than the variance, of this same value.

```
VarianceOfOutputUB[sys,t,1/8,True]
```

```
0.000219637
```

```
Plot[VarianceOfOutputUB[sys,t,tstar,True],
     {tstar, 0, 10},
   PlotLabel->"VarianceOfOutputUB[sys,t,tstar,True]",
   AxesLabel->{"tstar",""}]
```




```
- Graphics -
```

■ CDFVarianceOfOutputUB[cdf,x, t, tstar, max:1]

This function is the equivalent of VarianceOfOutputUB in the mixed case.

```
CDFVarianceOfOutputUB[cdftnormd,x,t,1/4]
```

```
0.000478785
```

■ UpperStatesProbability[sys,j]

This function will return the probability of being in a state greater than or equal to j. Note that this is not necessarily equal to 1-F[x] in the case of mixed systems.

```
UpperStatesProbability[sys,1/3] // Simplify
```

$$E^{-2\,t}\,(1+2\,t+2\,t^2)$$

```
(UpperStatesProbability[sys,#]& /@
   Transpose[sys][[1]]) // Simplify
```

$$\{1,\ E^{-2\,t}\,(1+2\,t+2\,t^2),\ E^{-2\,t}\,(1+2\,t),\ E^{-2\,t}\}$$

```
UpperStatesProbability[sys,1/3] /. t->1
```

$$\frac{5}{E^2}$$

■ CDFUpperStatesProbability[cdf,x,j]

This function is the equivalent of UpperStatesProbability for cdfs.

```
CDFUpperStatesProbability[cdfbeta,x,7/8] // N
```

```
0.40808
```

```
CDFUpperStatesProbability[cdfmult /. t->1,x,1/3]
```

$$\frac{5}{E^2}$$

■ StateDwellTime[sys,t,j,ni:False]

This function returns the time the system is expected to remain in the given state, assuming no cutoff time. This measure may diverge for repairable systems, and for the minimal state of non-repairable systems.

```
StateDwellTime[sys,t,1]
```

$$\frac{1}{2}$$

```
Prepend[(StateDwellTime[sys,t,#]& /@
   Transpose[sys][[1,{2,3,4}]]),Infinity]
```

$$\left\{\infty, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right\}$$

Note that, as expected, this measure diverges for the lowest state of this non-repairable system.

■ CDFStateDwellTime[cdf,x, j, t]

This function is the equivalent of StateDwellTime for CDFs.

```
CDFStateDwellTime[cdfmult,x,1,t]
```

```
0.5
```

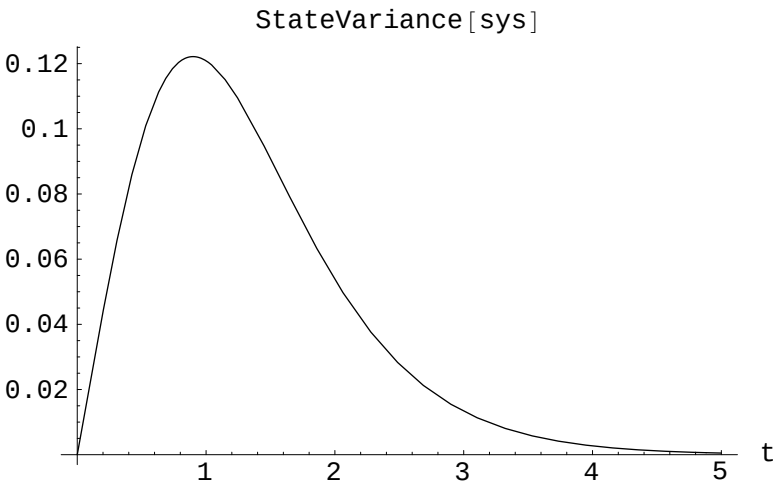
■ StateVariance[sys]

This function will return the variance of the system state.

```
StateVariance[sys] // Simplify
```

$$\frac{1}{9} E^{-4t} \left(9 \left(-1 + E^{2t}\right) + 8 \left(-3 + E^{2t}\right) t + 2 \left(-14 + E^{2t}\right) t^2 - 16 t^3 - 4 t^4\right)$$

```
Plot[StateVariance[sys], {t, 0, 5},
   PlotLabel->"StateVariance[sys]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

The customer may be interested in the point in time at which the system variance is at a maximum. This may be easily found:

```
t /. FindMinimum[-StateVariance[sys],{t,0}][[2]]
```

```
0.898822
```

```
StateVariance[sys /. t->1] // N
```

```
0.120867
```

■ CDFStateVariance[cdf,x, max:1]

This function is the equivalent of StateVariance for CDFs.

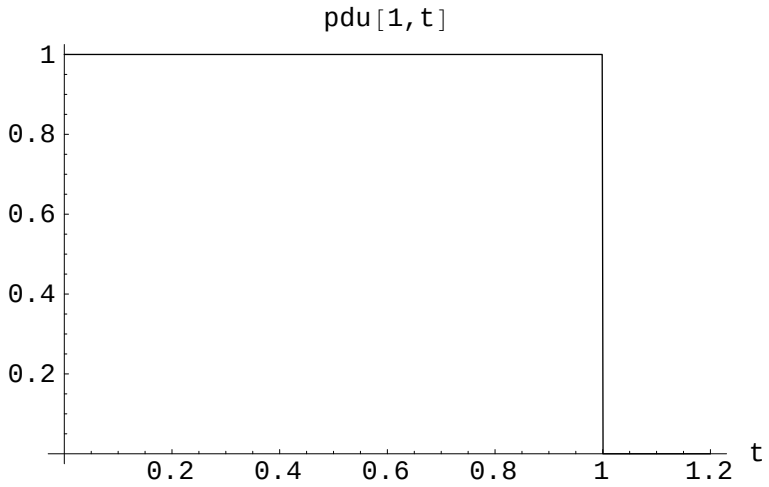
```
CDFStateVariance[cdfmult /. t->1, x]
```

```
0.120639
```

■ LifetimeWeighted[sys,t, u, utotal:1, ni:False]

This function returns the lifetime-weighted reliability measure, given sys and the time weighting function U[t]. utotal is the value of the integral of U[t] from 0 to Infinity. Note that lifetime weighting measures that expect this (non-repairable) system to perform well at large values of t produce smaller reliability measures, according to this scheme.

```
Plot[pdu[1,t], {t, 0, 1.2},
   PlotLabel->"pdu[1,t]",
   AxesLabel->{"t",""}]
```

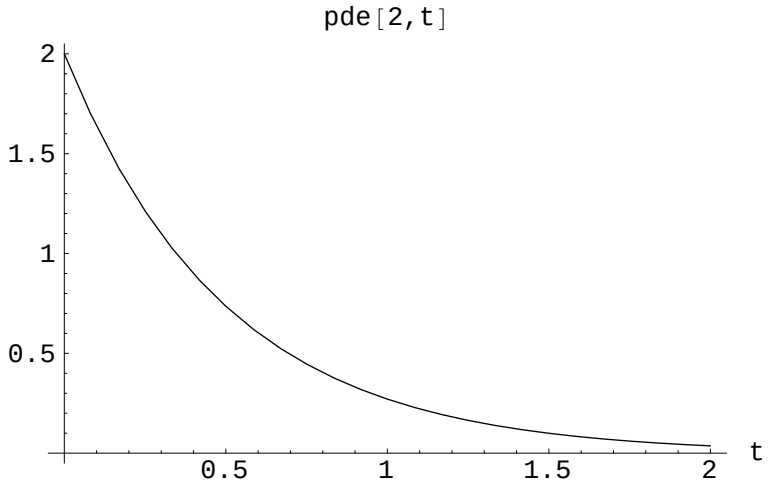



```
- Graphics -
```

```
LifetimeWeighted[sys, t, pdu[1,t], 1, True]
```

```
0.684218
```

```
Plot[pde[2,t], {t, 0, 2}, PlotLabel->"pde[2,t]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

```
LifetimeWeighted[sys, t, pde[2,t], 1] // N
```

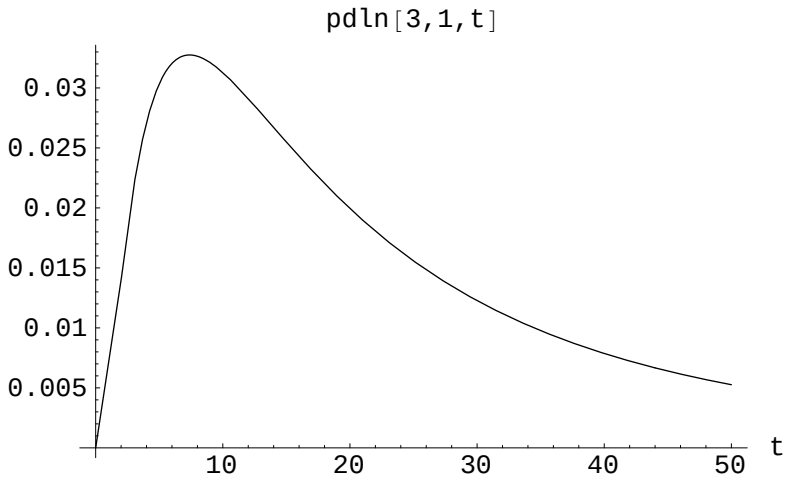
```
0.708333
```

In this case, a general solution may be obtained.

```
LifetimeWeighted[sys, t, pde[lambda,t], 1] // FullSimplify
```

$$\text{If}\left[\text{Re}[\text{lambda}] > -2,\ \frac{\text{lambda}\ (24 + 16\ \text{lambda} + 3\ \text{lambda}^2)}{3\ (2 + \text{lambda})^3},\ \int_0^\infty E^{-\text{lambda}\ t}\ \text{lambda}\ \left(E^{-2t} + \frac{4}{3} E^{-2t}\ t + \frac{2}{3} E^{-2t}\ t^2\right) dt\right]$$

```
Plot[pdln[3,1,t], {t, 0, 50}, PlotLabel->"pdln[3,1,t]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

```
LifetimeWeighted[sys, t, pdln[3,1,t], 1, True]
```

```
0.00414178
```

Note that our measure for Reliability (and hence customer satisfaction) is much lower for this weighting function. In general, the concept is that the weighting function U[t] should be proportional to the level of customer interest in the product (and hence, the degree to which performance of the product at that each point in time has the capacity to impact the customer's life and satisfaction with the product). The LogNormal curve above implies that the customer will be significantly affected by the behavior of the product at times far in the future. Since this system is non-repairable, and has low expected system states for large values of time, it receives a very small value of the lifetime weighting function under that scheme. The weighting functions Uniform and Exponential indicate (respectively) zero and very low levels of customer interest in the product at large values of t. The lifetime weigting functions are much greater for those values (note that the maximum is 1, so these measures indicate 70% of the customer satisfaction that would be attained if the product never left its maximal state). In general, any of the functions that begin with a lower case "p" that are defined in the Distributions' package would be good choices for U[t] functions, as they are all non-negative and are defined over the range [0, Infinity]. They will also all have a utotal of 1, so that need not be calculated. Additionally, the Lifetime Weighting measure has an additional interpretation in cases (such as this) when the U[t] function is the PDF of a random variable.

■ CDFLifetimeWeighted[cdf,x, t, u, utotal:1, max:1]

This function is the equivalent of LifetimeWeighted for CDFs.

```
CDFLifetimeWeighted[cdfbeta, x, t, pdu[1,t]]
```

```
0.833333
```

■ StateProbability[sys,j]

This function will return the probability of being in state j as a function of time.

```
(StateProbability[sys,#]& /@
   Transpose[sys][[1]]) // Simplify
```

$$\{E^{-2\,t}\,(-1+E^{2\,t}-2\,t-2\,t^2),\ 2\,E^{-2\,t}\,t^2,\ 2\,E^{-2\,t}\,t,\ E^{-2\,t}\}$$

```
StateProbability[sys /. t->1,1]
```

$$\frac{1}{E^2}$$

■ CDFStateProbability[cdf,x, x0:1]

This function is the equivalent of StateProbability for CDFs.

```
CDFStateProbability[cdfmult /. t->1, x]
```

$$\frac{1}{E^2}$$

■ UpperStatesDwellTime[sys,t,j,ni:False]

This function returns the time the system is expected to remain in states great than the given one, considering all time. See StateDwellTime for more information.

```
UpperStatesDwellTime[sys,t,3/4]
```

$$\frac{1}{2}$$

```
(UpperStatesDwellTime[sys,t,#]& /@
   Transpose[sys][[1]]) // Simplify
```

$$\left\{\frac{3}{2},\ 1,\ \frac{1}{2},\ 0\right\}$$

■ CDFUpperStatesDwellTime[cdf,x, j, t]

This function is the equivalent of UpperStatesDwellTime for CDFs.

```
CDFUpperStatesDwellTime[cdfmult, x, 3/4, t]
```

```
0.5
```

■ LowerStatesProbability[sys,j]

This function will return the probability of being in a state lower than the given one as a function of time. Note that this is NOT equal to the CDF.

```
(LowerStatesProbability[sys,#]& /@
   Transpose[sys][[1]]) // Simplify
```

$$\{0,\ E^{-2t}\ (-1+E^{2t}-2t-2t^2),\ E^{-2t}\ (-1+E^{2t}-2t),\ 1-E^{-2t}\}$$

```
LowerStatesProbability[sys,3/4]
```

$$1-E^{-2t}$$

■ CDFLowerStatesProbability[cdf,x, j]

This function will return the probability of being in a state lower than the given one as a function of time. Note that this is NOT equal to the CDF.

```
CDFLowerStatesProbability[cdfmult, x, 3/4]
```

$$1-E^{-2t}$$

■ RangeStatesProbability[sys,j, k]

This function will return the probability of being in any state below or equal to k, but above j, where j<k.

```
RangeStatesProbability[sys, 0, 2/3]
```

$$2E^{-2t}t+2E^{-2t}t^2$$

■ CDFRangeStatesProbability[cdf,x, j, k]

This function is the equivalent of RangeStatesProbabilityfor CDFs.

```
CDFRangeStatesProbability[cdfmult, x, 0, 2/3]
```

$$2\,E^{-2t}\,t + 2\,E^{-2t}\,t^2$$

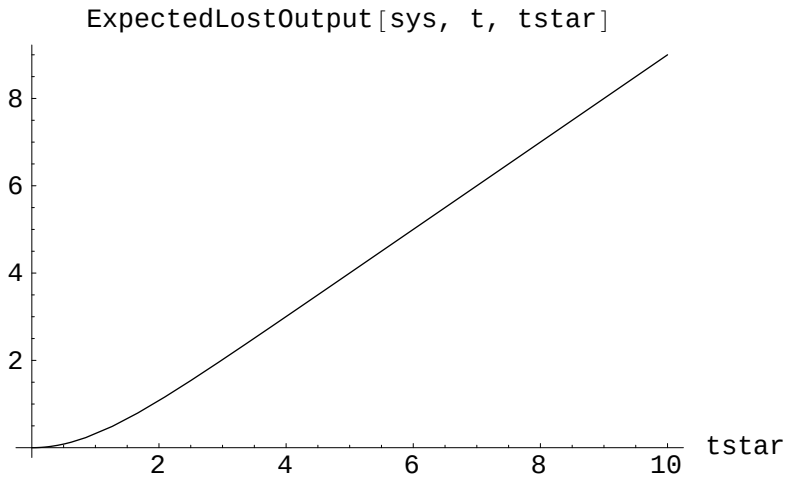
■ ExpectedLostOutput[sys,t, tstar, ni:False]

This function will return min*tstar-ExpectedOutput[sys,t,tstar].

```
ExpectedLostOutput[sys, t, tstar]
```

$$-1 + \text{tstar} + \frac{1}{3}\,E^{-2\,\text{tstar}}\,(3 + 3\,\text{tstar} + \text{tstar}^2)$$

```
Plot[ExpectedLostOutput[sys, t, tstar], {tstar, 0, 10},
   PlotLabel->"ExpectedLostOutput[sys, t, tstar]",
   AxesLabel->{"tstar",""}]
```



- Graphics -

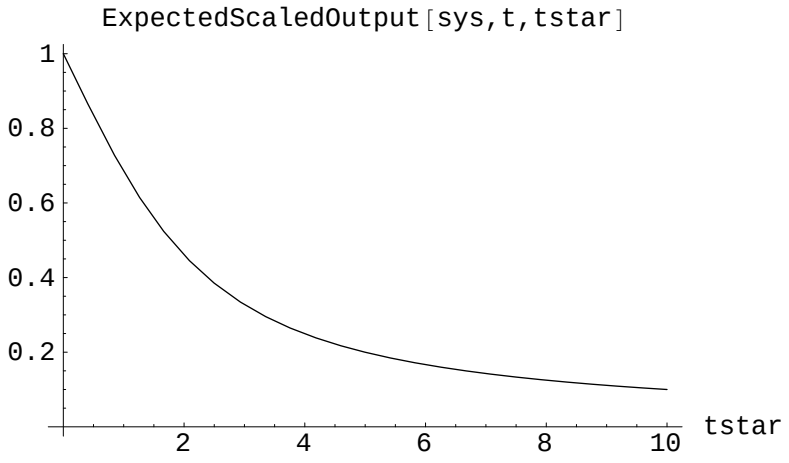
■ ExpectedScaledOutput[sys,t,tstar,ni:False]

This function will return ExpectedOutput[sys,t,tstar]/tstar.

```
ExpectedScaledOutput[sys,t,tstar]
```

$$\frac{1 - \frac{1}{3}\,E^{-2\,\text{tstar}}\,(3 + 3\,\text{tstar} + \text{tstar}^2)}{\text{tstar}}$$

```
Plot[ExpectedScaledOutput[sys,t,tstar], {tstar, 0, 10},
   PlotLabel->"ExpectedScaledOutput[sys,t,tstar]",
   AxesLabel->{"tstar",""}]
```




```
- Graphics -
```

■ CDFExpectedScaledOutput[cdfx, t, tstar, max:1]

This function is the equivalent of ExpectedScaledOutput for CDFs.

```
CDFExpectedScaledOutput[cdftnormd,x,t,1]
```

```
0.629473
```

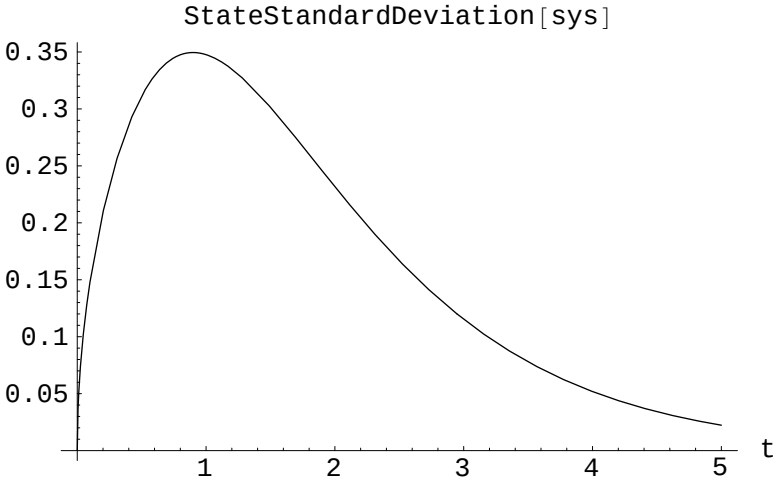
■ StateStandardDeviation[sys]

This function returns the standard deviation of the state of the system.

```
StateStandardDeviation[sys]
```

$$\sqrt{E^{-2t} + \frac{8}{9} E^{-2t} t + \frac{2}{9} E^{-2t} t^2 - \left(E^{-2t} + \frac{4}{3} E^{-2t} t + \frac{2}{3} E^{-2t} t^2\right)^2}$$

```
Plot[StateStandardDeviation[sys], {t, 0, 5},
   PlotLabel->"StateStandardDeviation[sys]",
   AxesLabel->{"t",""}]
```



- Graphics -

■ CDFStateStandardDeviation[cdf,x, max:1]

This function is the equivalent of CDFStateStandardDeviation for CDFs.

```
CDFStateStandardDeviation[cdfbeta,x]
```

```
0.103362
```

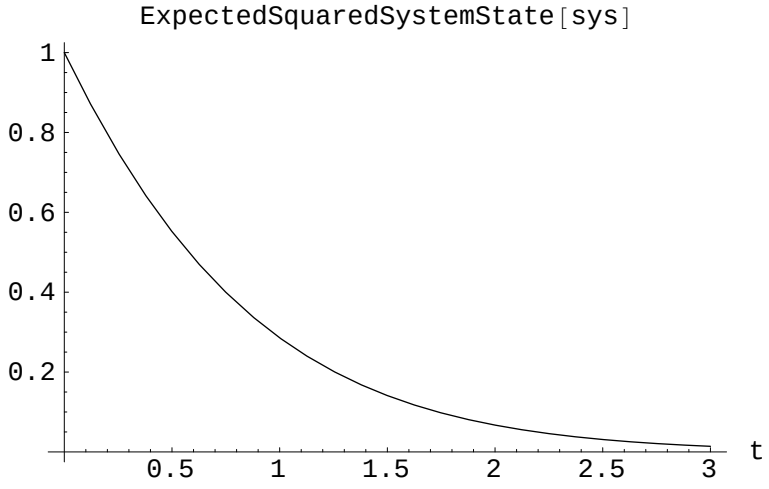
■ ExpectedSquaredSystemState[sys]

This function returns the expected value of the square of the system state. To calculate this for CDFs, use the function CDFMoment[cdf,x, 2, max].

```
ExpectedSquaredSystemState[sys]
```

$$E^{-2t} + \frac{8}{9} E^{-2t} t + \frac{2}{9} E^{-2t} t^2$$

```
Plot[ExpectedSquaredSystemState[sys], {t, 0, 3},
   PlotLabel->"ExpectedSquaredSystemState[sys]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

■ DerivativeOfLSP[sys,t, j]

This function returns the time derivative of the lower states probability function, for state j.

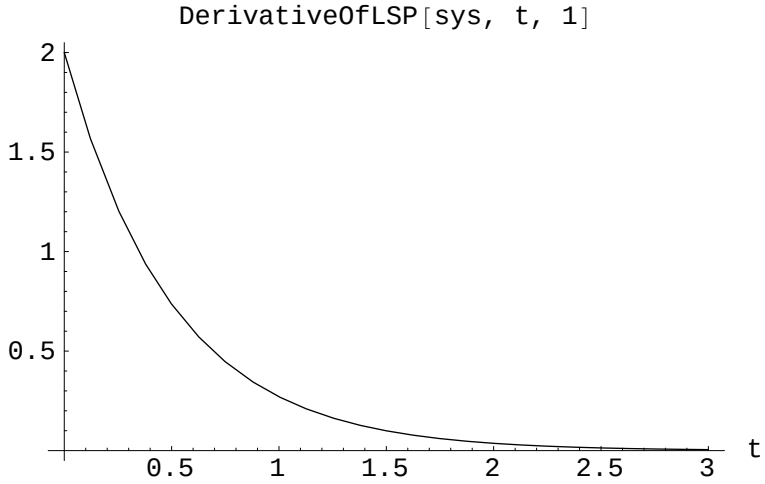
```
(DerivativeOfLSP[sys, t, #]& /@
   Transpose[sys][[1]]) // Simplify
```

$$\{0, 4\,E^{-2\,t}\,t^2, 4\,E^{-2\,t}\,t, 2\,E^{-2\,t}\}$$

```
DerivativeOfLSP[sys, t, 1]
```

$$2\,E^{-2\,t}$$

```
Plot[Evaluate[DerivativeOfLSP[sys, t, 1]], {t, 0, 3},
   PlotLabel->"DerivativeOfLSP[sys, t, 1]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

■ CDFDerivativeOfLSP[cdf,x, j, t]

This function is the equivalent of DerivativeOfLSP for CDFs.

```
CDFDerivativeOfLSP[cdfmult, x, 2/3, t]
```

$$4\,E^{-2\,t}\,t$$

■ DegradationRate[sys,t,j]

This function will return the rate of degradation from states higher than or equal to j to states lower than j. Please see also Hazard and CDFHazard, which may be of more general value than this function.

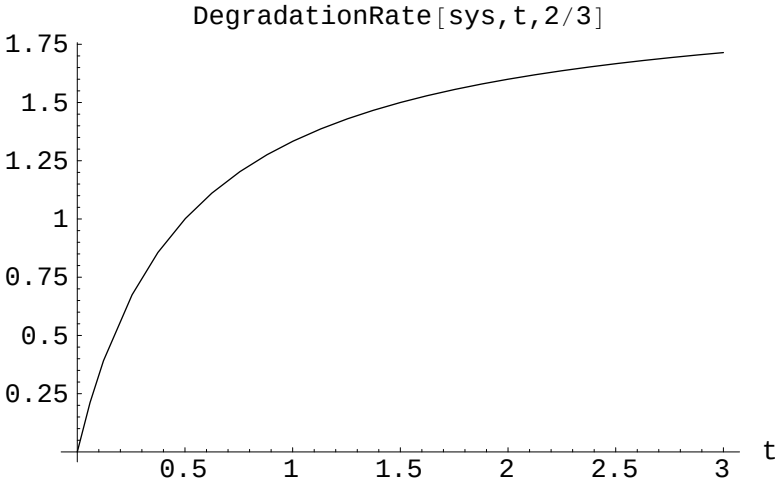
```
(DegradationRate[sys,t,#]& /@
   Transpose[sys][[1]]) // Simplify
```

$$\left\{0,\ \frac{4\,t^2}{1+2\,t+2\,t^2},\ \frac{4\,t}{1+2\,t},\ 2\right\}$$

```
DegradationRate[sys,t,2/3] // Simplify
```

$$\frac{4\,t}{1+2\,t}$$

```
Plot[Evaluate[DegradationRate[sys,t,2/3]], {t, 0, 3},
   PlotLabel->"DegradationRate[sys,t,2/3]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

■ Hazard[sys, t, j]

This function will return, for non-repairable systems and components, the rate of passing into or below state j given that the system is above state j at time t.

```
(Hazard[sys, t, #]& /@
   Transpose[sys][[1,{1,2,3}]]) // Simplify
```

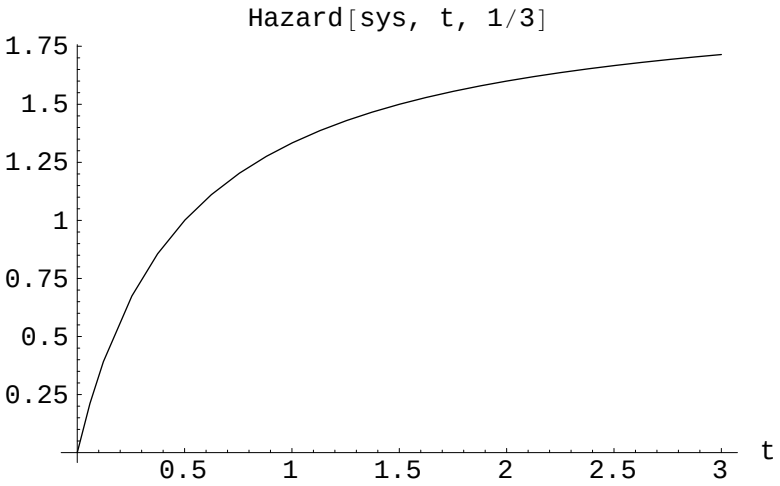
$$\left\{\frac{4\,t^2}{1+2\,t+2\,t^2},\ \frac{4\,t}{1+2\,t},\ 2\right\}$$

Note that this measure does not exist for the maximal state, as there is no state above the maximal state. Note that Hazard[sys,t, 1/3] = DegradationRate[sys,t,2/3].

```
Hazard[sys, t, 1/3] // Simplify
```

$$\frac{4\,t}{1+2\,t}$$

```
Plot[Evaluate[Hazard[sys, t, 1/3]], {t, 0, 3},
   PlotLabel->"Hazard[sys, t, 1/3]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

```
Hazard[sys, t, 1/3] /. t->1 // N
```

```
1.33333
```

■ CDFHazard[cdf,x, x0r, t, t0r, prec:$MachinePrecision]

This function is the equivalent of Hazard for CDFs. It is for level x0r of functioning and at time t0r. Greater precision can be obtained by increasing $MachinePrecision above its machine-dependent value (commonly 16). An alternative version of this function, CDFHazardB, is available, which uses Mathematica's own numerical differentiation routines. In all other respects it is the same. Tests seem to favor CDFHazard over CDFHazardB for all reliability problems attempted.

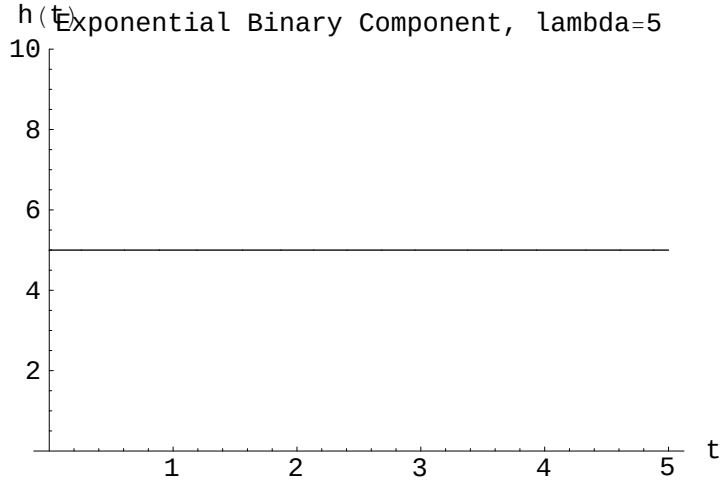
```
CDFHazard[cdfmult, x, 1/3, t, 1]
```

```
1.3333
```

This function will produce the ordinary binary Hazard function as a special case. To do this, x0r should be set to 0, and cdf should be a binary cdf where the probability of being in state 1 is equal to the probability of the system you are considering being in state 1.

```
hazsys=BinaryCDF[x,rde[5,t]];
```

```
Plot[CDFHazard[hazsys, x, 0, t, t0r],{t0r, 0, 5},
   PlotLabel->"Exponential Binary Component, lambda=5",
   AxesLabel->{"t","h(t)"},
   PlotRange->{0,10}]
```




```
- Graphics -
```

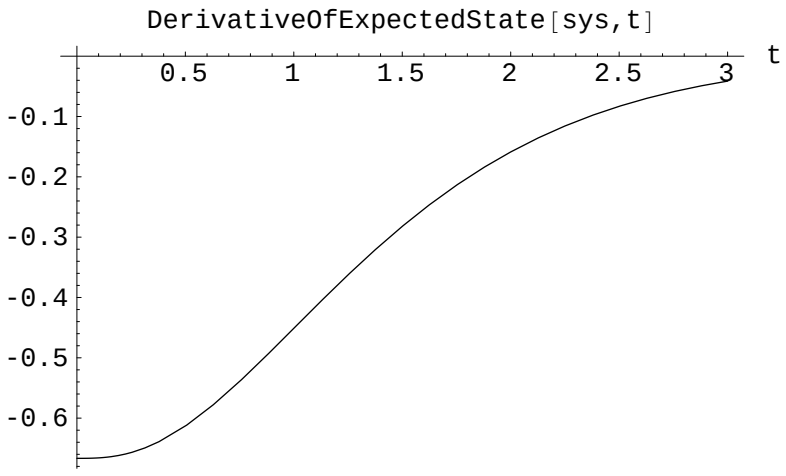
■ DerivativeOfExpectedState[sys,t]

This function returns the derivative of the expected state of the system.

```
DerivativeOfExpectedState[sys,t] // Simplify
```

$$-\frac{2}{3}\,E^{-2t}\,(1+2t+2t^2)$$

```
Plot[Evaluate[DerivativeOfExpectedState[sys,t]],
      {t, 0, 3},
   PlotLabel->"DerivativeOfExpectedState[sys,t]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

As one would anticipate, for this non-repairable system the rate of change of the system state approaches zero, as the probability that the system has been trapped in its minimal state is increasing.

```
DerivativeOfExpectedState[sys,t] /. t->1 // N
```

```
-0.451118
```

■ CDFDerivativeOfExpectedState[cdfx, t, t0r, prec:$MachinePrecision]

This function is the equivalent of DerivativeOfExpectedState for CDFs. See also CDFHazard. Like CDFHazard, this function performs numerical differentiation, which is much more rife with problems than numerical integration. It may be necessary to increase prec to obtain good results, as it was for CDFHazard.

```
CDFDerivativeOfExpectedState[cdfmult, x, t, 1]
```

```
-0.4620814841871379
```

■ CumulativeStandardDeviation[sys, t, tstar, ni:False]

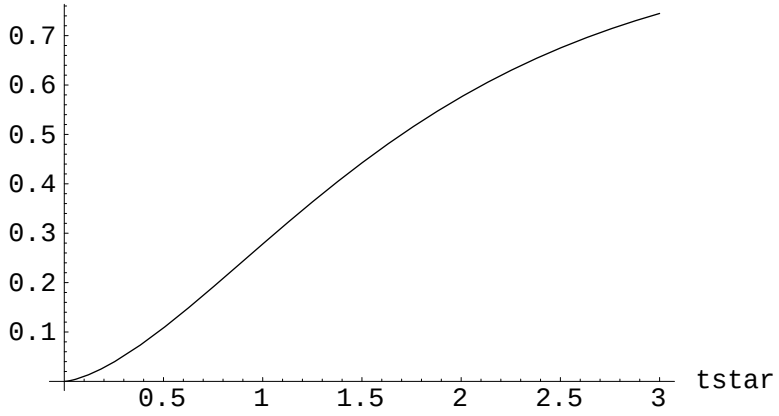
This function returns the integral over the system's useful lifetime of the standard deviation of its system state.

```
CumulativeStandardDeviation[sys, t, 2, True]
```

```
0.57588
```

```
Plot[CumulativeStandardDeviation[sys, t, tstar, True],
   {tstar, 0, 3},
   PlotLabel->
      "CumulativeStandardDeviation[sys,t,tstar,True]",
   AxesLabel->{"tstar",""}]
```

```
CumulativeStandardDeviation[sys,t,tstar,True]
```




```
- Graphics -
```

■ CDFCumulativeStandardDeviation[cdfx, t, tstar, max:1]

This function is the equivalent of CumulativeStandardDeviation for CDFs.

```
CDFCumulativeStandardDeviation[cdftnormd, x, t, 1/10]
```

```
0.00733575
```

■ SYSF[sys, j]

This function will return the probability of the system being in or below the state j, given sys.

```
SYSF[sys, 2/3] // Simplify
```

$$1 - E^{-2\,t}$$

■ CDFF[cdf, x, x0:0]

This function is the equivalent of SYSF for CDFs.

```
CDFF[cdfmult, x, 2/3]
```

$$1 - E^{-2\,t}$$

■ SYSR[sys, j]

This function will return the probability of the system being above the state j.

```
SYSR[sys, 2/3]
```

$$E^{-2\,t}$$

■ CDFR[cdf, x, x0:0]

This function is the equivalent of SYSR for CDFs.

```
CDFR[cdfmult, x, 2/3]
```

$$E^{-2\,t}$$

■ ChebyshevUB[x,var, epsilon]

This function will return the Chebyshev Upper Bound. var is the variance of the random variable in question, mu is its mean. The expression returned is the upper bound of the probability that the random variable in question is epsilon or more units away from its mean. When only an upper bound of var is available (as in VarianceOfOutputUB), this inequality is still valied when that bound is used as input. The variable x is not used by this function.

```
ChebyshevUB[1, 1, 2]
```

$$\frac{1}{4}$$

■ Moment[sys,n:1]

This function will calculate the expected value of X^n.

```
Moment[sys,6] // Simplify
```

$$\frac{1}{729} E^{-2t} (729 + 128\, t + 2\, t^2)$$

■ CDFMoment[cdf,x,n:1,max:1]

This function is the equivalent of Moment for CDFs. It is the nth moment (with respect to the origin) of a random variable x given its cdf. It is assumed that this random variable is non-negative.

```
CDFMoment[cdftnormd /. t->1,x,2]
```

```
0.145335
```

■ CDFMoment[cdf,x,n:1,max:1]

This function is the equivalent of Moment for CDFs. It is the nth moment (with respect to the origin) of a random variable x given its cdf. It is assumed that this random variable is non-negative.

```
CDFMoment[cdftnormd /. t->1,x,2]
```

```
0.145335
```

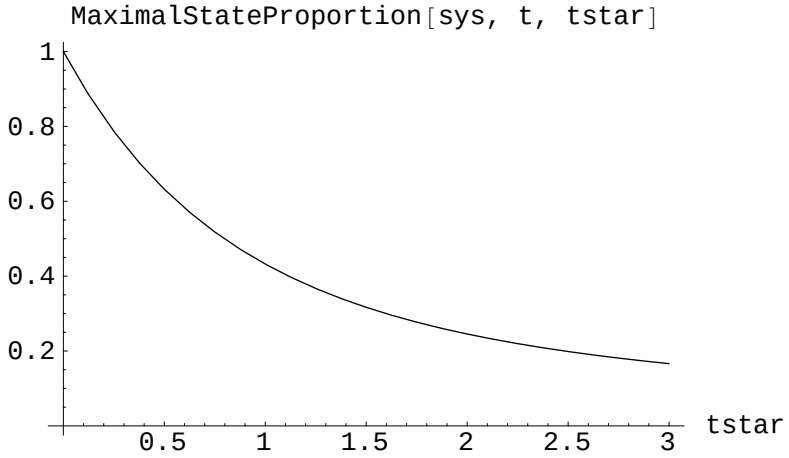
■ MaximalStateProportion[sys,t, tstar, ni:False]

This function will find the expression for the integral of probability of the system being in its maximal state, over the system's useful lifetime and divided by the useful lifetime.

```
MaximalStateProportion[sys, t, tstar]
```

$$\frac{1}{2\,\text{tstar}} - \frac{E^{-2\,\text{tstar}}}{2\,\text{tstar}}$$

```
Plot[MaximalStateProportion[sys, t, tstar],
   {tstar, 0, 3},
   PlotLabel->"MaximalStateProportion[sys, t, tstar]",
   AxesLabel->{"tstar",""}]
```




```
- Graphics -
```

```
MaximalStateProportion[sys, t, tstar] /. tstar -> 1/8 // N
```

```
0.884797
```

■ CDFMaximalStateProportion[cdfx, t, tstar, max:1]

This function will find the expression for the integral of probability of the system being in its maximal state, over the system's useful lifetime and divided by the useful lifetime.

```
CDFMaximalStateProportion[cdfmult, x, t, 1/8]
```

```
0.884797
```

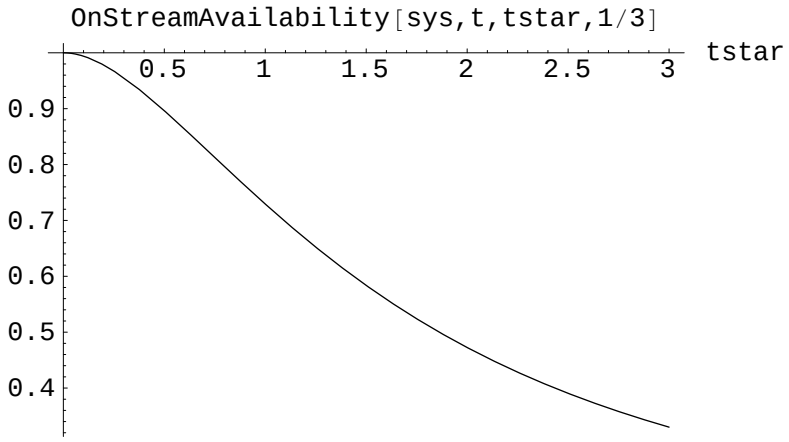
■ OnStreamAvailability[sys,t, tstar, j, ni:False]

This function will return the integral of the probability of the system being above state j, over the system's useful lifetime tstar and divided by that useful lifetime.

```
OnStreamAvailability[sys, t, tstar, 1/3]
```

$$\frac{1}{\text{tstar}} - \frac{E^{-2\,\text{tstar}}\,(1+\text{tstar})}{\text{tstar}}$$

```
Plot[OnStreamAvailability[sys,t,tstar,1/3],
   {tstar, 0, 3},
   PlotLabel->"OnStreamAvailability[sys,t,tstar,1/3]",
   AxesLabel->{"tstar",""}]
```




```
- Graphics -
```

```
OnStreamAvailability[sys, t, tstar, 1/3] /. tstar->1 // N
```

```
0.729329
```

■ CDFOnStreamAvailability[cdf,x, j, t, tstar]

This function is the equivalent of OnStreamAvailability for CDFs.

```
CDFOnStreamAvailability[cdfmult, x, 1/3, t, 1]
```

```
0.729329
```

■ DiscreteEntropy[sys]

This function will return the calculated entropy of the given multistate sys configuration. See also ContinuousEntropy.

```
DiscreteEntropy[sys /. t->1]
```

$$\frac{2}{E^2} - \frac{4\,\text{Log}\left[\frac{2}{E^2}\right]}{E^2} - \frac{(-5+E^2)\,\text{Log}\left[\frac{-5+E^2}{E^2}\right]}{E^2}$$

```
DiscreteEntropy[sys /. t->1] // N
```

```
1.34319
```

■ ContinuousEntropy[f,x, min:0, max:1, ni:False]

This function will return the calculated entropy of the given PDF (note: NOT CDF) over the given domain. The domain should not include sets where f(x)=0.

```
ContinuousEntropy[pdfbeta, x]
```

$$\frac{103619}{27720} - \text{Log}[110]$$

```
ContinuousEntropy[pdfbeta, x] // N
```

```
-0.962421
```

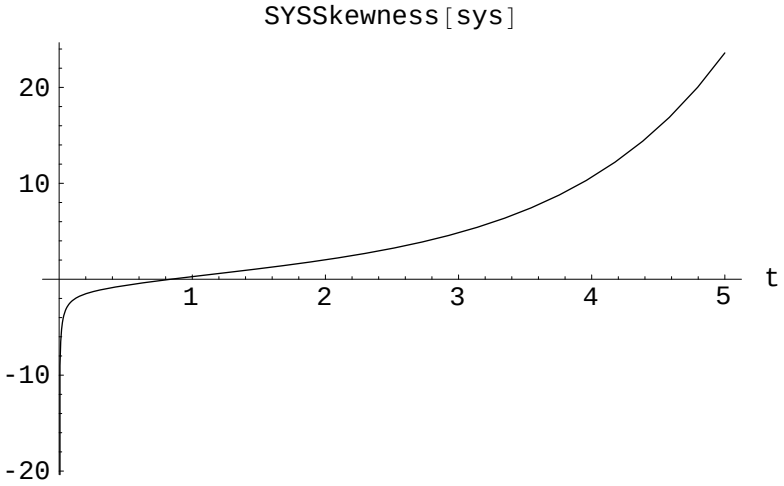
■ SYSSkewness[sys]

This function will return the skewness of the given system.

```
s1=(SYSSkewness[sys] // Simplify)
```

$$\left(E^{-6t}\left(27E^{4t}+16E^{4t}t+2E^{4t}t^2+2(3+4t+2t^2)^3-3E^{2t}(3+4t+2t^2)(9+8t+2t^2)\right)\right)\Big/\left(E^{-4t}(9(-1+E^{2t})+8(-3+E^{2t})t+2(-14+E^{2t})t^2-16t^3-4t^4)\right)^{3/2}$$

```
Plot[s1, {t, 0, 5}, PlotLabel->"SYSSkewness[sys]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

■ CDFSkewness[cdf,x, max:1]

This function is the equivalent of SYSSkewness for CDFs. For this function and others of its type (typically where max is an option, but not min), the CDF is assumed to be non-negative.

```
CDFSkewness[cdfbeta, x]
```

```
-0.921401
```

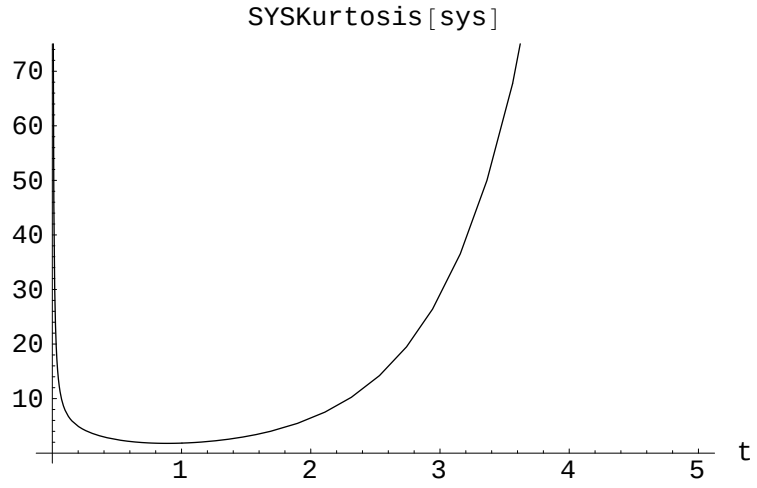
■ SYSKurtosis[sys]

This function will return the kurtosis of the system.

```
k1 = (SYSKurtosis[sys] // Simplify)
```

$$\frac{\left(81\left(-3+6E^{2t}-4E^{4t}+E^{6t}\right)+16\left(-81+108E^{2t}-39E^{4t}+2E^{6t}\right)t+2\left(-1620+1386E^{2t}-248E^{4t}+E^{6t}\right)t^2-32\left(153-78E^{2t}+5E^{4t}\right)t^3-8\left(609-165E^{2t}+2E^{4t}\right)t^4+192\left(-17+2E^{2t}\right)t^5+48\left(-30+E^{2t}\right)t^6-384t^7-48t^8\right)}{\left(9\left(-1+E^{2t}\right)+8\left(-3+E^{2t}\right)t+2\left(-14+E^{2t}\right)t^2-16t^3-4t^4\right)^2}$$

```
Plot[k1, {t, 0, 5}, PlotLabel->"SYSKurtosis[sys]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

■ CDFKurtosis[cdf,x, max:1]

This function is the equivalent of SYSKurtosis for CDFs.

```
CDFKurtosis[cdfbeta, x]
```

```
3.78857
```

■ SYSKurtosisExcess[sys]

This function is the KurtosisExcess of the given system. It is just SYSKurtosis-3, so see SYSKurtosis for a graph of its behavior for this system.

```
SYSKurtosisExcess[sys /. t->1] // N
```

```
-1.15932
```

■ CDFKurtosisExcess[cdf,x, max:1]

This function is the equivalent of SYSKurtosisExcess for CDFs.

```
CDFKurtosisExcess[cdfbeta, x]
```

```
0.788571
```

■ SYSQuantile[sys,q:1/2]

This function will return the qth quantile of the multistate configuration sys.

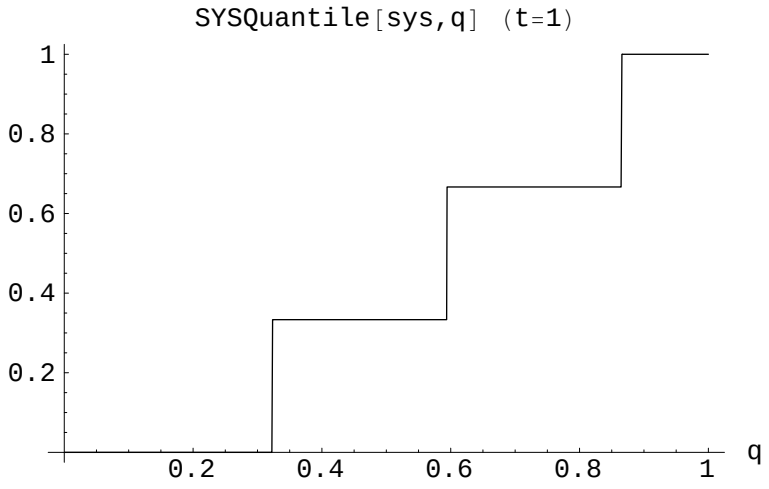
```
SYSQuantile[N[sys /. t->1], 2/3]
```

```
0.666667
```

```
sys /. t->1 // N
```

```
{{0, 0.323324}, {0.333333, 0.270671}, {0.666667, 0.270671},
 {1., 0.135335}}
```

```
Plot[SYSQuantile[N[sys /. t->1], q], {q, 0, 1},
   PlotLabel->"SYSQuantile[sys,q] (t=1)",
   AxesLabel->{"q",""}]
```




```
- Graphics -
```

■ CDFQuantile[cdf,x, q:1/2, prec$MachinePrecision,min:0, max:1]

This function will return the qth quantile of the x. There are two additional functions, CDFQuantileUp and CDFQuantileDown, that are related to CDFQuantile. These aren't intended to be called directly, but instead take multiple points that meet the quantile criteria as the supremum and infinimum of that set, respectively, for use in certain other functions.

```
CDFQuantile[cdfbeta, x, 2/3]
```

```
0.892792
```

■ SYSMedian[sys]

This function will return the median of the SYS system.

```
SYSMedian[N[sys /. t->1]]
```

```
0.333333
```

■ CDFMedian[cdf,x, min:0, max:1]

This function will return the median of the CDF system.

```
CDFMedian[cdfbeta, x]
```

```
0.852037
```

■ SYSQuartiles[sys]

This function will return the quartiles of the SYS system.

```
SYSQuartiles[N[sys /. t->1]]
```

```
{0, 0.333333 , 0.666667 }
```

■ CDFQuartiles[cdf,x, min:0, max:1]

This function will return the quartiles of the CDF system.

```
CDFQuartiles[cdfbeta, x]
```

```
{0.773373 , 0.852037 , 0.91239 }
```

■ SYSQuadraticRaw[sys,y0, k:1]

This function will return a sys multistate matrix with the states replaced by k (y-y0)^2. Once this system is constructed, a variety of measures such as ExpectedState, StateVariance, SYSSkewness, SYSKurtosis, and SYSKurtosisExcessmay be calculated (these are all separate measures for the CDF case, which is more complicated). Note that k (y-y0)^2 is a loss function (disutility). By using k=1 and y0=max, when min=0, this will return the raw configuration which would be used to calculate Boedigheimer's second measure (see Boedigheimer 1992, p. 170).

```
du=SYSQuadraticRaw[sys, 1]
```

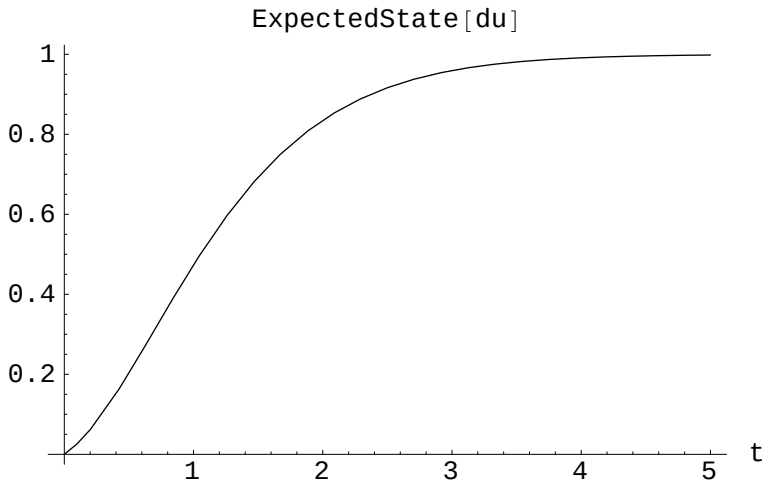
$$\left\{\{1,\ E^{-2\,t}\ (-1+E^{2\,t}-2\,t-2\,t^2)\},\ \left\{\frac{4}{9},\ 2\,E^{-2\,t}\,t^2\right\},\ \left\{\frac{1}{9},\ 2\,E^{-2\,t}\,t\right\},\ \{0,\ E^{-2\,t}\}\right\}$$

Here are a few measures of this disutility:

```
ExpectedState[du] // Simplify
```

$$\frac{1}{9}\,E^{-2\,t}\,(9\,(-1+E^{2\,t})-16\,t-10\,t^2)$$

```
Plot[ExpectedState[du], {t, 0, 5},
   PlotLabel->"ExpectedState[du]",
   AxesLabel->{"t",""}]
```

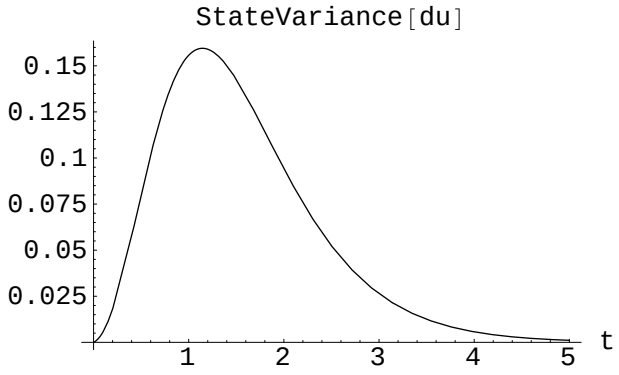



```
- Graphics -
```

```
StateVariance[du] // Simplify
```

$$\frac{1}{81} E^{-4t} \left(81 \left(-1 + E^{2t}\right) + 32 \left(-9 + 4 E^{2t}\right) t + \left(-436 + 50 E^{2t}\right) t^2 - 320 t^3 - 100 t^4\right)$$

```
Plot[StateVariance[du], {t, 0, 5},
   PlotLabel->"StateVariance[du]",
   AxesLabel->{"t",""}]
```




```
- Graphics -
```

A customer might be interested in learning when the uncertainty in the disutility associated with this product is at a maximum. That can be determined quite easily.

```
t /. FindMinimum[-StateVariance[du], {t, 0}][[2]]
```

```
1.14451
```

■ CDFQuadraticMean[cdf,x, y0, k:1, max:1]

This function will return the mean of the random variable where the state considered is the function k (y-y0)^2 of the ordinary states. Note that this is a quadratic loss function (disutility) such as that commonly used in quality studies. A variety of measures of this quadratic loss function will be detailed here. All contain "Quadratic" in their names, and the type of measure they represent should be self explanatory.

```
CDFQuadraticMean[cdfbeta, x, 1]
```

```
0.0384615
```

■ CDFQuadraticVariance[cdf,x, y0, k:1, max:1]

This function will return the variance of the quadratic loss function.

```
CDFQuadraticVariance[cdfbeta, x, 1]
```

```
0.00218371
```

■ CDFQuadraticSkewness[cdf,x, y0, k:1, max:1]

This function will return the skewness of the quadratic loss function.

```
CDFQuadraticSkewness[cdfbeta, x, 1]
```

```
2.51602
```

■ CDFQuadraticKurtosis[cdf,x, y0, k:1, max:1]

This function will return the kurtosis of the quadratic loss function.

```
CDFQuadraticKurtosis[cdfbeta, x, 1]
```

```
12.1641
```

■ CDFQuadraticKurtosisExcess[cdf,x, y0, k:1, max:1]

This function will return the kurtosis excess of the quadratic loss function.

```
CDFQuadraticKurtosisExcess[cdfbeta, x, 1]
```

```
9.16412
```

■ CDFQuadraticKurtosisExcess[cdf,x, y0, k:1, max:1]

This function will return the kurtosis excess of the quadratic loss function.

```
CDFQuadraticKurtosisExcess[cdfbeta, x, 1]
```

```
9.16412
```

■ SYSInterquartileRange[sys]

This function will return the interquartile range of the sys multistate matrix.

```
SYSInterquartileRange[N[sys /. t->1]]
```

```
0.666667
```

■ CDFInterquartileRange[cdf,x, min:0, max:1]

This function is the equivalent of SYSInterquartileRange for CDFs.

```
CDFInterquartileRange[cdfbeta, x]
```

```
0.139018
```

■ SYSQuartileDeviation[sys]

This function will return the quartile deviation of the multistate configuration sys.

```
SYSQuartileDeviation[N[sys /. t->1]]
```

```
0.333333
```

■ CDFQuartileDeviation[cdf,x, min:0, max:1]

This function is the equivalent of SYSQuartileDeviation for CDFs.

```
CDFQuartileDeviation[cdfbeta, x]
```

```
0.0695088
```

■ SYSPearsonSkewness2[sys]

This function will return the PearsonSkewnessII of a SYS system.

```
SYSPearsonSkewness2[N[sys /. t->1]]
```

```
0.627101
```

■ CDFPearsonSkewness2[cdf,x, min:0, max:1]

This function is the equivalent of SYSPearsonSkewness2 for CDFs.

```
CDFPearsonSkewness2[cdfbeta, x] // N
```

```
-0.542845
```

■ CDFRandom[cdf,x, n:1, min:0, max:1]

This function will return a random sample of size n from the distribution given by CDF[x], which has a domain that is some subset of [min, max].

```
CDFRandom[cdfbeta, x, 10]
```

```
{0.9428785062243037 , 0.7812975148991086 , 0.8322748273326965 ,
 0.9387823629323861 , 0.9212966710524029 , 0.8975155394676335 ,
 0.8982863206030629 , 0.8518726301001038 , 0.8084652550441547 ,
 0.8403898284857974 }
```

The ContinuousOptimization Package

■ General Comments and Definitions

This package provides a series of functions that can assist in the creation and optimization of structure functions for continuous systems. Scattered data interpolation techniques for continuous structure function estimation, and functions to determine optimal multistate discretization, are included.

Many of these functions are best served by having their functionality illustrated in context. Extended examples of the use of many of these functions are provided in special sections at the end of the function definitions section.

■ Function Documentation

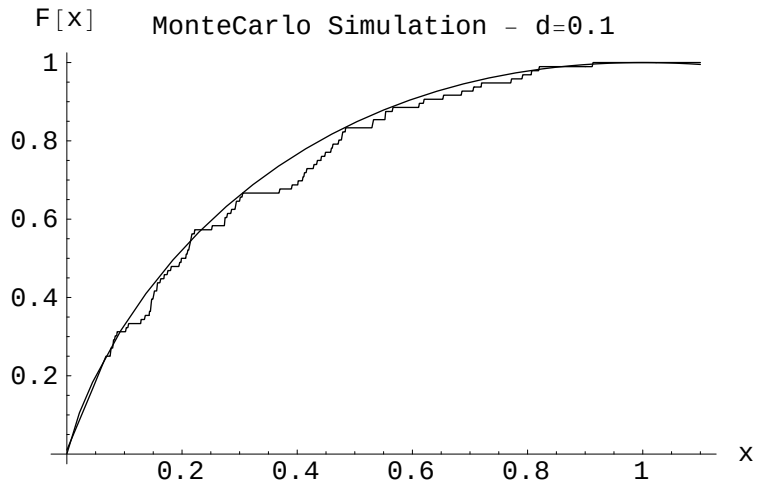
■ MonteCarlo2[cdflist,x, phi, d:0.1, min:0, max:1]

This function will return the MultistateCDF for a system, based on a list of CDF'sfor the components (cdflist). x is the independent variable in cdflist, phi is the function relating the component state vectors to the system state, and d is a value such that p +- d is a 95% confidence interval on the probability in question. It is assumed that the distributions have a minimum value of min and a maximum value of max. The method used is MonteCarlc simulation.

We will demonstrate this technique (and the one to follow) with a system for which an exact solution for the CDF is known.

```
phi$Pi[x_] := Times @@ x

cdflist={UniformCDF[x],UniformCDF[x]};

a1=MonteCarlo2[cdflist,x,phi$Pi,0.1];
```

```
Plot[Evaluate[{a1, x (1-Log[x])}], {x, 0.0001,1.1},
   PlotLabel->"MonteCarlo Simulation - d=0.1",
   AxesLabel->{"x", "F[x]"}]
```

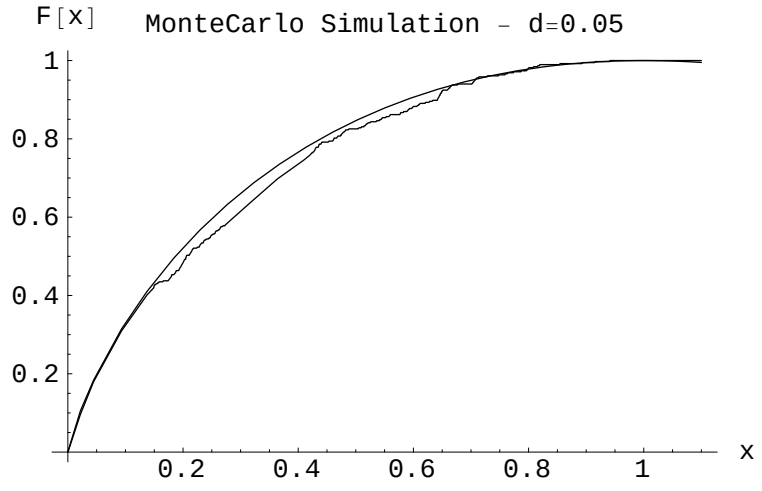



```
- Graphics -
```

Here is another approximation,this time with more data points.

```
a2=MonteCarlo2[cdflist,x,phi$Pi,0.05];
```

```
Plot[Evaluate[{a2, x (1-Log[x])}], {x, 0.0001,1.1},
   PlotLabel->"MonteCarlo Simulation - d=0.05",
   AxesLabel->{"x","F[x]"}]
```



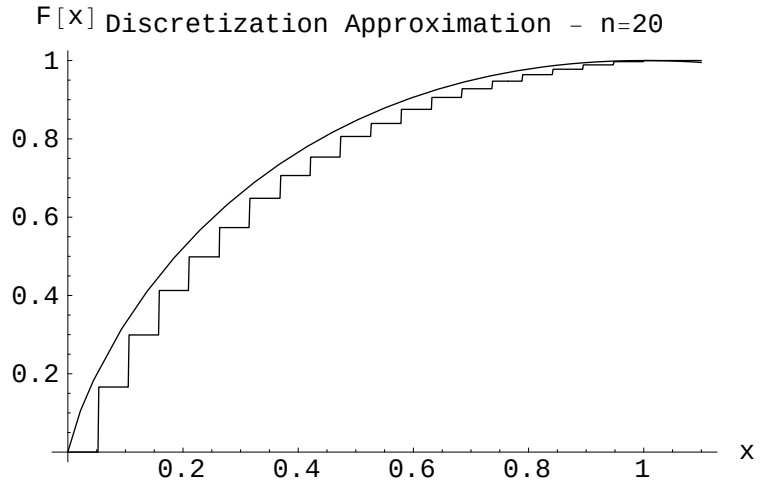

```
- Graphics -
```

■ ContDisc[cdflist,x,phi,n:5,min:0,max:1]

This function will calculate the Multistate CDF for the system, based on the list of CDF's for the components (cdflist). x is the independent variable in cdflist, phi is the function relating the components to the system, and n is the number of discrete points the system and components should be discretized into. It is assumed that the distributions have a minimum value of min and a maximum value of max. The technique used is discretization, as suggested by Montero [1990] (though this author intended this technique to be used for structural analysis, rather than stochastic analysis).

```
a3=MultistateCDF[x,ContDisc[cdflist,x,phi$Pi,20]];
```

```
Plot[Evaluate[{a3, x (1-Log[x])}], {x,0.0001,1.1},
   PlotLabel->"Discretization Approximation - n=20",
   AxesLabel->{"x","F[x]"}]
```




```
- Graphics -
```

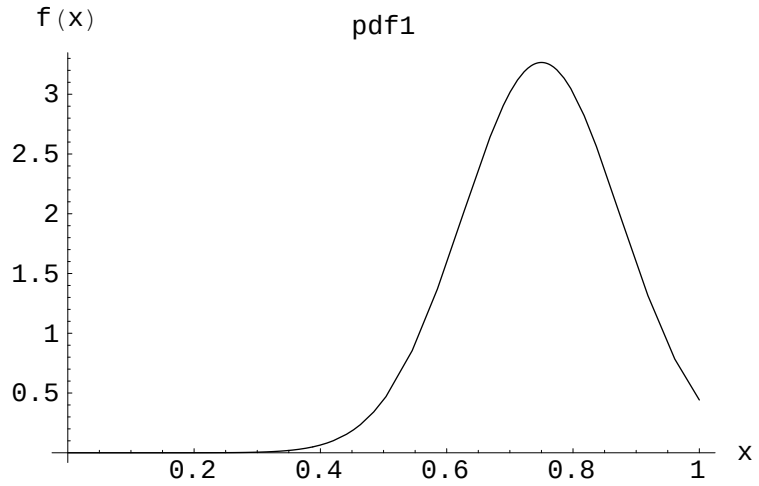
■ TruncatedPDF[x,mmadist,min:0,max:1]

This function will return the value of the PDF of a truncated mmadist distribution, truncated to lie between min and max. The function that we use below to illustrate TruncatedPDF will be used to illustrate other functions later in this chapter.

```
pdf1=TruncatedPDF[x,NormalDistribution[3/4,1/8]]
```

$$\frac{4\,E^{-32\left(-\frac{3}{4}+x\right)^2}\sqrt{\frac{2}{\pi}}}{\frac{1}{2}\left(1+\text{Erf}\left[\sqrt{2}\right]\right)+\frac{1}{2}\left(-1+\text{Erf}\left[3\sqrt{2}\right]\right)}$$

```
Plot[pdf1,{x,0,1},PlotLabel->"pdf1",
   AxesLabel->{"x","f(x)"}]
```




```
- Graphics -
```

■ PDFADist[pdf,x,a]

This function will return the RMS deviation from the discrete values for a continuous component or system, given the PDF for that component or system. The independent variable for this distribution is x, and a is the list of boundary values for the different discrete values. The assumption here, as follows into the related functions such as RD, GenAns, etc., is that the continuous values between the first two boundary points are mapped to the first boundary point, that continuous values between the last two boundary points are mapped to the last boundary point, and that other adjacent pairs of boundary points map their values to a point midway between them. Thus, the extrema of the continuous model are retained.

```
PDFADist[pdf1,x,{0,1/3,2/3,1}]
```

```
0.197518
```

Please see the "DetermininingMultistate Discretizations"section for more examples of the use of this function.

■ RD[pdf,x,n,min:0,max:1]

This function will find the optimal values for the boundaries of the different levels for a continuous system discretized into n distinct states, based on minimum values of PDFADist for the entire distribution. The distribution is given by pdf, and the independent variable is x. It is assumed that the minimal value for the PDF is min, and the maximal value is max.

```
RD03[pdf1,x]
```

```
{0.111183 , {0, 0.463156 , 0.856707 , 1}}
```

Please see the "DetermininingMultistate Discretizations"section for more examples of the use of this function.

■ GenAns[pdf,x,a]

This function will return a table (in the normal form in which multistate sytems are given) for the given list of boundaries, a, and the given pdf with independent variable x.

```
GenAns[pdf1,x,{0,1/3,2/3,1}]
```

$$\left\{\{0,\ 0.000439048\ \},\ \left\{\frac{1}{2},\ 0.257931\right\},\ \{1,\ 0.74163\ \}\right\}$$

Please see the "DetermininingMultistate Discretizations"section for more examples of the use of this function.

■ CohInputQ[data]

This function will check the given input data set to see if it is non-decreasing. If it is, True is returned (False if not). If the set is not coherent, pairs of numbers are printed to the screen that indicate pairs of components (by their order in the original data set) that violate the non-decreasing requirement.

```
inp = {{{0,0},0},{{0,1/2},1/2},{{1/2,0},1/2},
       {{1/2,1/2},3/4},{{1,1},1}}
```

$$\left\{\{\{0, 0\}, 0\}, \left\{\{0, \frac{1}{2}\}, \frac{1}{2}\right\}, \left\{\{\frac{1}{2}, 0\}, \frac{1}{2}\right\}, \left\{\{\frac{1}{2}, \frac{1}{2}\}, \frac{3}{4}\right\}, \{\{1, 1\}, 1\}\right\}$$

```
CohInputQ[inp]
```

```
True
```

Please see the "Performing Scattered and Gridded Data Interpolations" section for more examples of the use of this function.

■ ExtremaAdd[data]

This function will check if the zero vector maps to 0 and the one vector maps to 1 in the given data set, and adds them if they are not.

```
ExtremaAdd[inp]
```

$$\left\{\{\{0, 0\}, 0\}, \left\{\{0, \frac{1}{2}\}, \frac{1}{2}\right\}, \left\{\{\frac{1}{2}, 0\}, \frac{1}{2}\right\}, \left\{\{\frac{1}{2}, \frac{1}{2}\}, \frac{3}{4}\right\}, \{\{1, 1\}, 1\}\right\}$$

Please see the "Performing Scattered and Gridded Data Interpolations" section for more examples of the use of this function.

■ Shepard[x,data]

This function will calculate the approximation to the phi value at the point x, given the input data set data. If the extreme values are not present, they are not added. If the point x is equal to one of the data points, then the phi value is equal to the data point's phi value at that data point. The method used is Shepard's method. Shepard's method exhibits the property that the value returned for a given point is greater than or equal to the minimum of all the given data points, and less than or equal to the maximum of all the given data points.

```
Shepard[{1/3,1/3},inp] // N
```

```
0.573964
```

Please see the "Performing Scattered and Gridded Data Interpolations" section for more examples of the use of this function.

■ MultiQuadric[x,data,c,rsq:(1/6),prec:$MachinePrecision]

This function will calculate the approximation to the phi value at the point x, given the input data set data and the constant vector c (calculated by MultiQuadricC. If the extreme values are not present, they are not added. The method used is Hardy's MultiQuadric method, which is considered to be a radial basis function method. In general, the best choice of rsq depends almost exclusively on the data function values, rather than the distribution of the data sites. It is also nearly indpendent of the number of data points. rsq should be smaller for rapidly varying data sets, and might also need to be small if there are a very large number of data points to prevent the matrix used to solve for c from being ill-conditioned. Tarwater [1985] showed that the RMS error obtained when using this method decreases as rsq increases, up to some optimal value; beyond this value, errors often increase dramatically as the associated linear system of equations becomes ill-conditioned. As rsq approaches zero, the solution becomes tighter, as it increases, the solution becomes smoother. An examination of RMSError values for random continuous systems with between 10 and 100 data points, using either Min or Max structure functions, and with either 2 or three components, indicated that optimal values of rsq for this type of problem are generally close to 1/6; in no case was the optimal value of rsq greater than 1/2.

```
MultiQuadric[{1/3,1/3},inp,MultiQuadricC[inp]]
```

```
0.5486315773404066
```

Please see the "Performing Scattered and Gridded Data Interpolations" section for more examples of the use of this function, and please see the "Determining Optimal RSQ Values" section for a discussion of selecting optimal rsq values for this function and all others that accept rsq as a parameter.

■ MultiQuadricC[data,rsq:(1/6),prec:$MachinePrecision]

This function will calculate the constant vector needed for the MultiQuadric routine.

```
MultiQuadricC[inp]
```

```
{2.03356, -0.80133, -0.80133, -0.241413, 0.272716}
```

Please see the "Performing Scattered and Gridded Data Interpolations" section for more examples of the use of this function.

■ ShepardND[data,m:5]

This function will return a data set consisting of approximations of phi values at data points on a gridded set with mesh density m. The extrema points are added to the customer-supplied data set data, if they are not already present. If there are any customer-supplied data points other than the minimal or maximal ones which have minimal or maximal phi values (respectively), then any mesh points less then (or greater than) these are given minimal (or maximal) values. The gridded points are calculated in ascending order of Euclidian distance from the midpoint of the component space hypercube, so that central values are calculated first. After each point is calculated, it is compared against all points less than or greater than it so that the non-decreasing property of coherent systems is not violated, and the minimal or maximal values allowed are assumed should the Shepard estimate violate the non-decreasing property as originally calculated. As each point is calculated, it is added to the generating data set so that subsequent calculations may use it as part of the knowledge base about the system. Note that one potential modification of this approach would be to only consider the data points that whose sites are strictly less than or strictly greater than the given one, and then depend on properties of Shepard's rule to insure that the resulting grid is non-decreasing. This approach was rejected due to the possibility of some data sets being very sparse, and so basing estimatations on very distant points. Also, although two adjacent but non-ordered points are not required have any order relation between their function values, in practice nearby points are often close in functional value.

```
ShepardND[inp] // N[#,2]& // MatrixForm[#,TableSpacing->{0}]&
```

{0.5, 0.5}	0.75
{0.25, 0.5}	0.58
{0.5, 0.25}	0.58
{0.5, 0.75}	0.75
{0.75, 0.5}	0.75
{0.25, 0.25}	0.53
{0.25, 0.75}	0.63
{0.75, 0.25}	0.63
{0.75, 0.75}	0.75
{0, 0.5}	0.5
{0.5, 0}	0.5
{0.5, 1.}	0.75
{1., 0.5}	0.75
{0, 0.25}	0.46
{0, 0.75}	0.59
{0.25, 0}	0.46
{0.25, 1.}	0.66
{0.75, 0}	0.59
{0.75, 1.}	0.75
{1., 0.25}	0.66
{1., 0.75}	0.75
{0, 0}	0
{0, 1.}	0.63
{1., 0}	0.63
{1., 1.}	1.

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

■ MultiQuadricND[data,m:5,rsq:(1/6),prec:$MachinePrecision]

This function will return a data set consisting of approximations of phi values at data points on a gridded set with mesh density m. The extrema points are added to the customer-supplied data set data, if they are not already present. If there are any customer-supplied data points other than the minimal or maximal ones which have minimal or maximal phi values (respectively), then any mesh points less then (or greater than) these are given minimal (or maximal) values. The gridded points are calculated in ascending order of Euclidian distance from the midpoint of the component space hypercube, so that central values are calculated first. After each point is calculated, it is compared against all points less than or greater than it so that the non-decreasing property of coherent systems is not violated, and the minimal or maximal values allowed are assumed should the MultiQuadric estimate violate the non-decreasing property as originally calculated. As each point is calculated, it is added to the generating data set so that subsequent calculations may use it as part of the knowledge base about the system.

```
MultiQuadricND[inp] // N[#,2]& //
   MatrixForm[#,TableSpacing->{0}]&
```

{0.5, 0.5}	0.75
{0.25, 0.5}	0.62
{0.5, 0.25}	0.62
{0.5, 0.75}	0.85
{0.75, 0.5}	0.85
{0.25, 0.25}	0.41
{0.25, 0.75}	0.78
{0.75, 0.25}	0.78
{0.75, 0.75}	0.9
{0, 0.5}	0.5
{0.5, 0}	0.5
{0.5, 1.}	0.92
{1., 0.5}	0.92
{0, 0.25}	0.23
{0, 0.75}	0.7
{0.25, 0}	0.23
{0.25, 1.}	0.88
{0.75, 0}	0.7
{0.75, 1.}	0.96
{1., 0.25}	0.88
{1., 0.75}	0.96
{0, 0}	0
{0, 1.}	0.82
{1., 0}	0.82
{1., 1.}	1.

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

■ MLinInt[x,data]

This function will approximate the value of phi[x], given the gridded input data in the set data (usual form). It uses a linear interpolation.

```
mlans = PhiGrid[Max[#]&,20,2];
```

```
MLinInt[{1/8,1/8},mlans] // N
```

```
0.137336
```

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

■ GriddedRMSError[data,phi]

This function will estimate the RMS error term of an approximation given by the data set data (which is in the same form as the data sets which normally contain input data for the interpolation functions) as compared to the true function phi. The squared differences from the true values at all points on the grid are summed, this total is divided by the number of grid points, and the square root of this entire expression is returned. The extrema are NOT added.

```
mlans2 = PhiGrid[Max[#]&,5,2];
```

```
GriddedRMSError[mlans2,Max[#]&]
```

```
0
```

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

■ ShepardRMSError[data,phi,m:10]

This function will estimate the RMS error term of a Shepard approximation over a grid of mesh density m, as compared to the true function phi. The squared differences from the true values at all points on the grid are summed, this total is divided by the number of grid points, and the square root of this entire expression is returned. The input data points are taken as the set data, and the extrema are NOT added.

```
phi7[x_] := Max[x]
```

```
test7=ExtremaAdd[RandomPhi[phi7,16]];
```

```
ShepardRMSError[test7,phi7]
```

```
0.0867636
```

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function. Please note that your result for ShepardRMSError will probably be different if you recalculate this notebook, as RandomPhi is a random function.

■ MultiQuadricRMSError[data,phi,m:10,rsq:(1/6), prec:$MachinePrecision]

This function will estimate the RMS error term of a MultiQuadric approximation over a grid of mesh density m, as compared to the true function phi. The squared differences from the true values at all points on the grid are summed, this total is divided by the number of grid points, and the square root of this entire expression is returned. The input data points are taken as the set data, and the extrema are NOT added. See MultiQuadric for a discussion of the meaning of the rsq variable, and of prec.

```
MultiQuadricRMSError[test7,phi7]
```

```
0.025369
```

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function. Please note that your result for MultiQuadricRMSError will probably be different if you recalculate this notebook, as RandomPhi is a random function.

■ PhiGrid[phi,m:4,n:2]

This function will return a set of data points, where each is of the form {{x11,x12,...,x1n},phi1}. The points specified by the x coordinates are those comprising a mesh of density m, and the phii values are determined by the function phi. Note that there will be m^n data points, rather than m.

```
PhiGrid[Max[#]&,3,2]
```

$$\left\{\{\{0, 0\}, 0\}, \left\{\left\{0, \frac{1}{2}\right\}, \frac{1}{2}\right\}, \{\{0, 1\}, 1\}, \left\{\left\{\frac{1}{2}, 0\right\}, \frac{1}{2}\right\}, \left\{\left\{\frac{1}{2}, \frac{1}{2}\right\}, \frac{1}{2}\right\}, \left\{\left\{\frac{1}{2}, 1\right\}, 1\right\}, \{\{1, 0\}, 1\}, \left\{\left\{1, \frac{1}{2}\right\}, 1\right\}, \{\{1, 1\}, 1\}\right\}$$

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

■ RandomPhi[phi,m:4,n:2]

This function will return a set of m data points, where each is of the form {{x11,x12,... ,x1n},phi1}. The points specified by the x coordinates are generated randomly, and the phii values are determined by the function phi.

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

```
RandomPhi[Max[#]&,3,2]
```

```
{{{0.900006 , 0.264095 }, 0.900006 },
 {{0.524465 , 0.278477 }, 0.524465 },
 {{0.202887 , 0.640141 }, 0.640141 }}
```

■ RandomGenerate[m:4,n:2]

This function will return a set of m sets of n random numbers (uniformly distributed).

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

```
RandomGenerate[3,2]
```

```
{{0.197711 , 0.397095 }, {0.86322 , 0.14914 }, {0.239146 , 0.0174086 }}
```

■ ShepardGrid[data,m:4]

This function will, given a set of scattered data points (in the form {{{x11,x12,...,x1n},phi1},...}) over the component state space, return a regular grid of mesh density m over that space, where the structure function value at each point is estimated with Shepard'sMethod. The value returned is in the same form as data. The assumption is made and utilized that the zero vectors maps to 0, and the one vector maps to 1.

```
ShepardGrid[inp] // N[#,2]& //
   MatrixForm[#,TableSpacing->{0}]&
```

{0, 0}	0
{0, 0.33}	0.44
{0, 0.67}	0.51
{0, 1.}	0.56
{0.33, 0}	0.44
{0.33, 0.33}	0.57
{0.33, 0.67}	0.65
{0.33, 1.}	0.65
{0.67, 0}	0.51
{0.67, 0.33}	0.65
{0.67, 0.67}	0.72
{0.67, 1.}	0.82
{1., 0}	0.56
{1., 0.33}	0.65
{1., 0.67}	0.82
{1., 1.}	1.

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

■ GridGenerate[m:4,n:2]

This function will return a list containing all the points that ma ke up an evenly spaced grid spanning the full range of each of the n components. Each component will be reduced to m discrete values in the grid.

```
GridGenerate[3,2]
```

$$\left\{\{0, 0\}, \left\{0, \frac{1}{2}\right\}, \{0, 1\}, \left\{\frac{1}{2}, 0\right\}, \left\{\frac{1}{2}, \frac{1}{2}\right\}, \left\{\frac{1}{2}, 1\right\}, \{1, 0\}, \left\{1, \frac{1}{2}\right\}, \{1, 1\}\right\}$$

Please see the "Performing Scattered and Gridded Data Interpolations" section for examples of the use of this function.

■ Determining Multistate Discretizations

Although this material was in essence covered in the function documentation, it merits further consideration and examples.

In essence, we use the function PDFADist to assess the quality of any particular discretization, when the true distribution is known. We use the function RD to obtain an optimal discretization. Both will be illustrated here.

Please refer to the beginning of this chapter for an illustration and definition of pdf1, the density function that will be used extensively in this section. It will be taken to represent the PDF for the state of the component that we wish to discretize

We examine various discretizations for this component below, noting that while discretizations with more levels produce better results, large differences in results can be obtained with the same number of levels by shifting the positioning of the boundaries between the levels. It appears that in this case better results are obtained by having more different levels available in regions where the component is likely to reside:

```
PDFADist[pdf1,x,{0,1/2,1}]
```

```
0.27871
```

```
PDFADist[pdf1,x,{0,1/4,1/2,3/4,1}]
```

```
0.130594
```

```
PDFADist[pdf1,x,{0,3/8,5/8,7/8,1}]
```

```
0.0732353
```

```
PDFADist[pdf1,x,{0,3/8,1/2,5/8,3/4,7/8,1}]
```

```
0.0474665
```

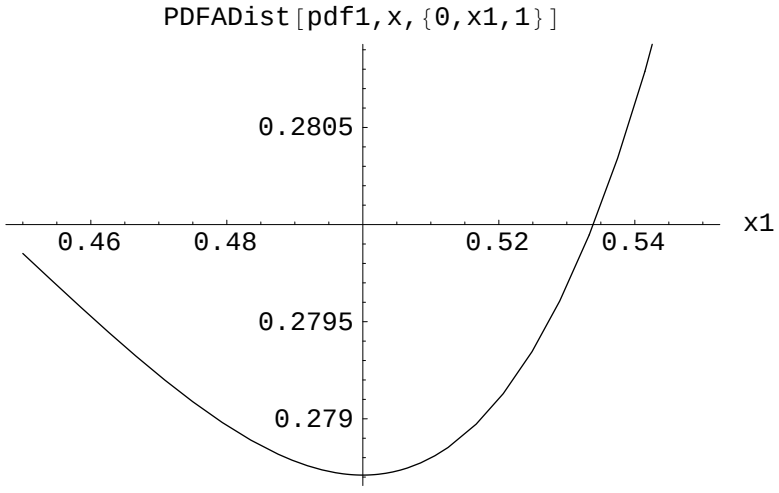
Now, we examine the optimal discretization for a two-level discretization. Please note that RD calls the Mathematica function FindMinimum, which attempts to find only a LOCAL minimum. However, examinations of graphs showed that (in all cases considered) it was finding the global minimum. However, if this is a concern a global optimization routine could be substituted for the local one.

```
ans2=RD[pdf1,x,2]
```

```
{0.27871, {0, 0.5, 1}}
```

Here is a graph of the quality of the two-level discretization, as a function of where the boundary between the two levels was placed:

```
Plot[PDFADist[pdf1,x,{0,x1,1}],{x1,0.45,0.55},
   PlotLabel->"PDFADist[pdf1,x,{0,x1,1}]",
   AxesLabel->{"x1",""}]
```



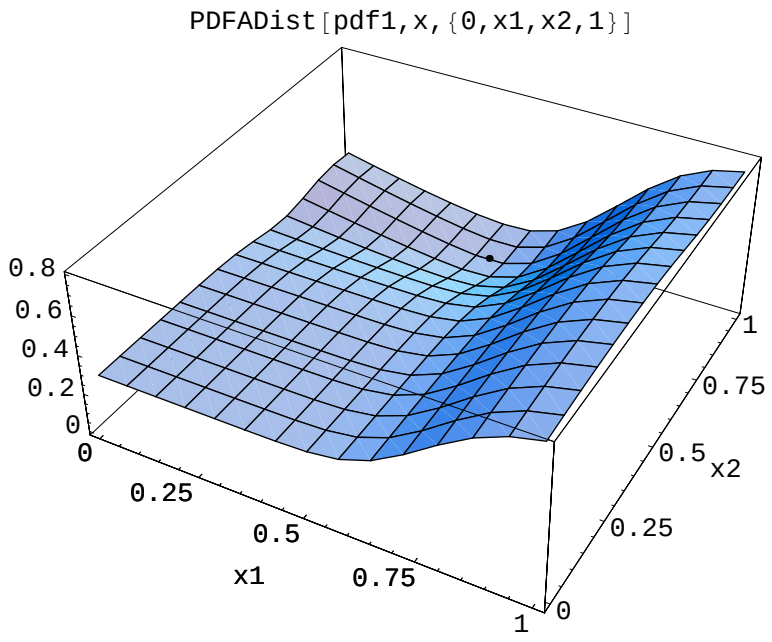

```
- Graphics -
```

Similarly, here is the same analyis for a three-level discretization. The black dot on the graph is the optimal point found with the RD function below.

```
ans3=RD[pdf1,x,3]
```

```
{0.111183, {0, 0.463156, 0.856707, 1}}
```

```
Show[Plot3D[PDFADist[pdf1,x,{0,x1,x2,1}],
     {x1,0,1},{x2,0,1},
            PlotLabel->"PDFADist[pdf1,x,{0,x1,x2,1}]",
            AxesLabel->{"x1","x2",""},
            DisplayFunction->Identity],
     Graphics3D[Point[{ans3[[2,2]],ans3[[2,3]],
                       ans3[[1]]+0.01}],
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction]
```




```
- Graphics3D -
```

Now, we ask for all optimal discretizations for numbers of levels ranging from two to nine

```
ansi = Table[RD[pdf1,x,i],{i,2,9}];
```

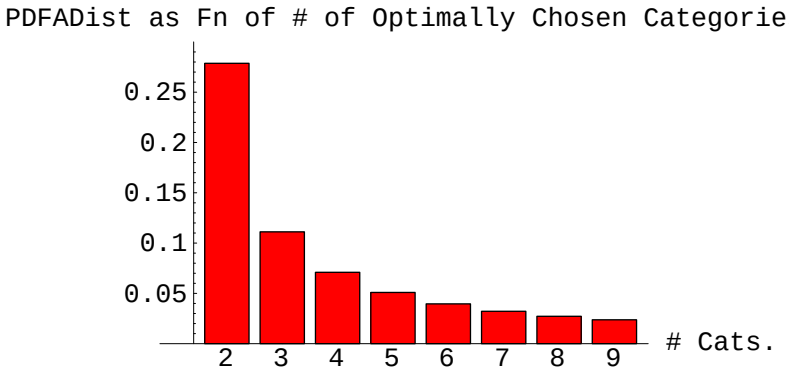
```
ansi
```

```
{{0.27871, {0, 0.5, 1}}, {0.111183, {0, 0.463156, 0.856707, 1}},
 {0.0709472, {0, 0.413886, 0.667966, 0.88286, 1}}, {0.0509273,
  {0, 0.38246, 0.59957, 0.753086, 0.907553, 1}}, {0.0395751,
  {0, 0.357868, 0.555424, 0.680731, 0.799709, 0.926197, 1}},
 {0.0322012, {0, 0.341005,
   0.527339, 0.644081, 0.73758, 0.829989, 0.934915, 1}},
 {0.0272168, {0, 0.324791, 0.497115,
   0.61303, 0.693553, 0.77111, 0.851415, 0.944778, 1}},
 {0.0237478, {0, 0.303779, 0.462548, 0.584152, 0.66529,
   0.733302, 0.799706, 0.869726, 0.951824, 1}}}
```

Here is a chart of how the RMS error goes down as the number of levels increases:

```
<<Graphics‘Graphics‘
```

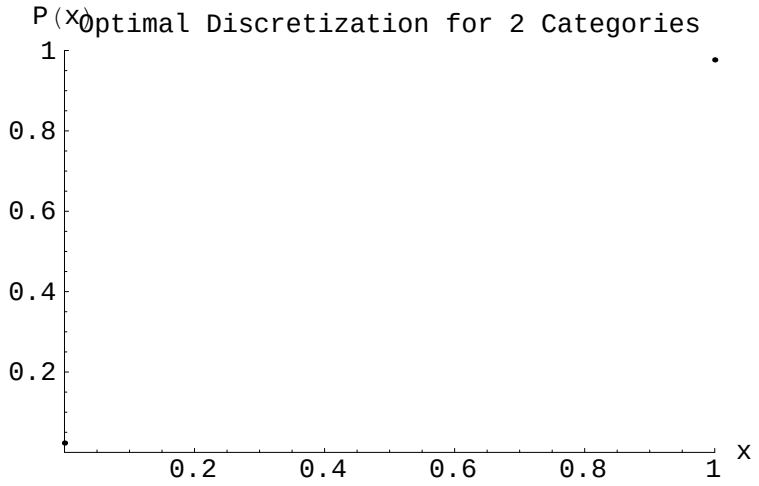
```
BarChart[Transpose[ansi][[1]], BarLabels->Range[2,9],
 PlotLabel->
  "PDFADist as Fn of # of Optimally Chosen Categories",
 AxesLabel->{"# Cats.",""},
 PlotRange->{0,0.30}]
```

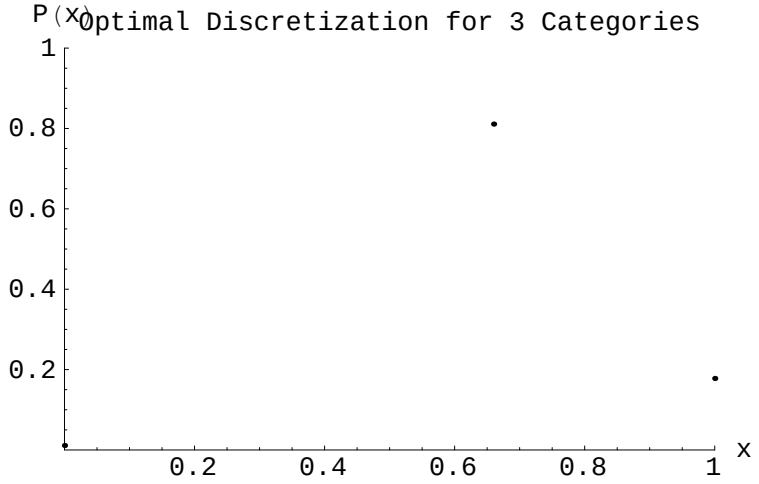


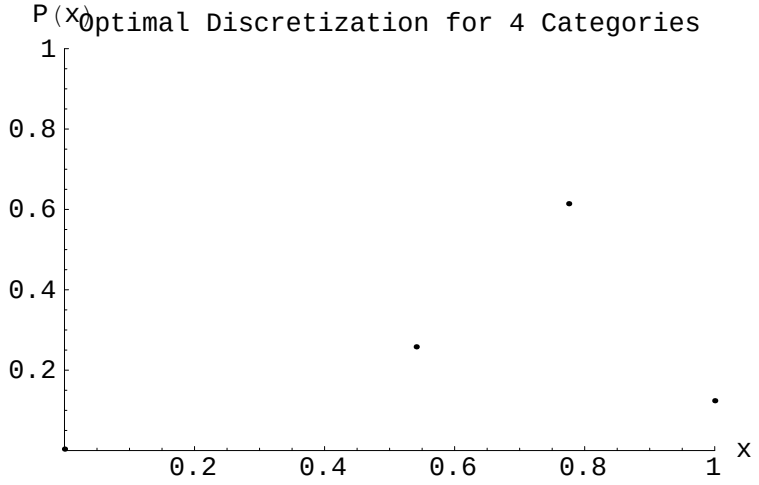

```
- Graphics -
```

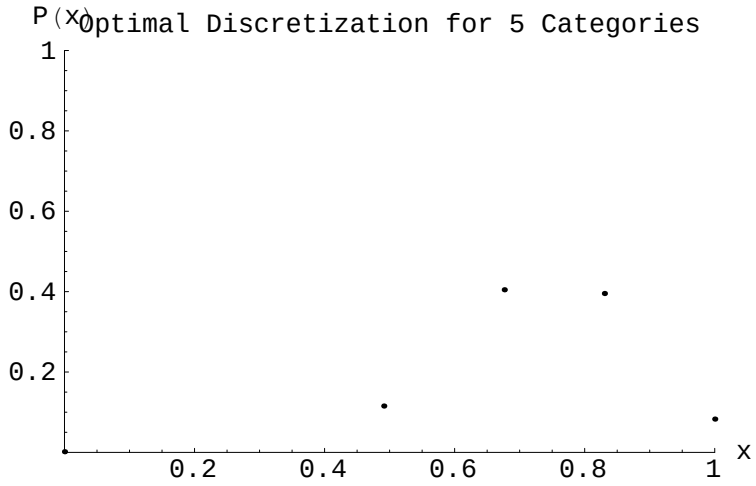
Finally, we examine the multistate system graphs for these discretizations. For each point on these graphs, the x-value represents a discrete point whose value the discretized component may assume, the the y-value for that point represents the probability that the component will take on that value. Note that the plots, as the number of levels increases, approaches in appearance the original PDF.

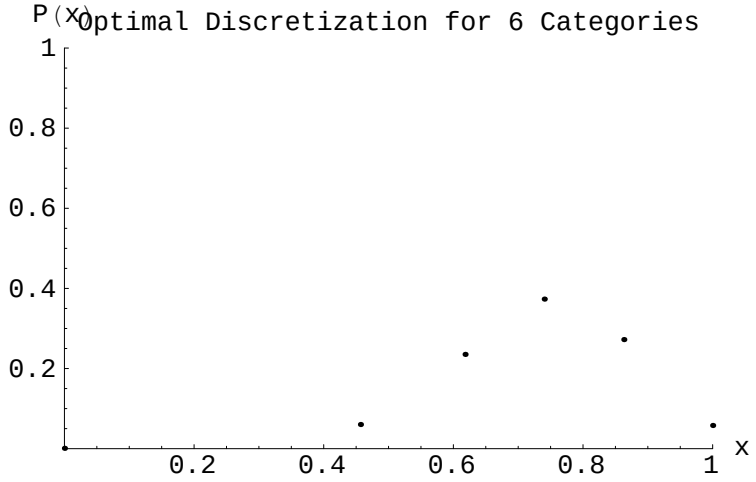
```
PlotAns[pdf_,x_,a_] :=
   ListPlot[GenAns[pdf,x,a],
            PlotRange->{{0,1},{0,1}},
            PlotLabel->
"Optimal Discretization for " <>
   ToString[Length[a]-1] <>" Categories",
            AxesLabel->{"x","P(x)"},
            DisplayFunction->Identity]

allplots = PlotAns[pdf1,x,#[[2]]]& /@ ansi;

Do[Show[allplots[[i]],
     DisplayFunction->$DisplayFunction],
   {i,Length[allplots]}]
```

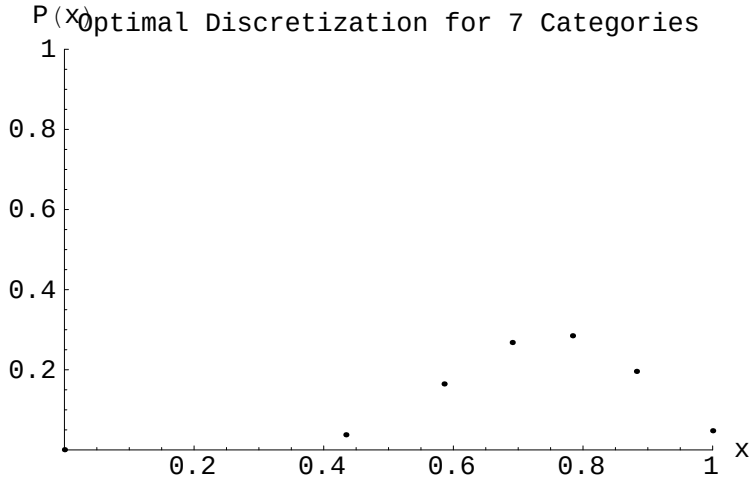

P(x)
Optimal Discretization for 2 Categories
1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1
x

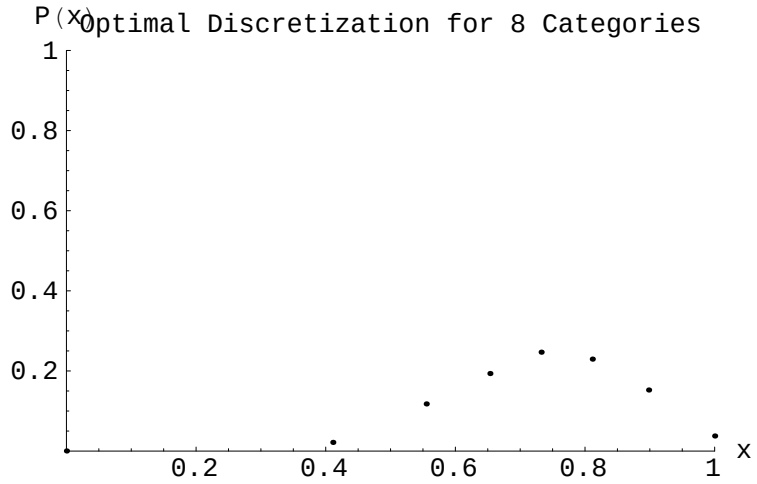

P(x)
Optimal Discretization for 3 Categories
1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1
x

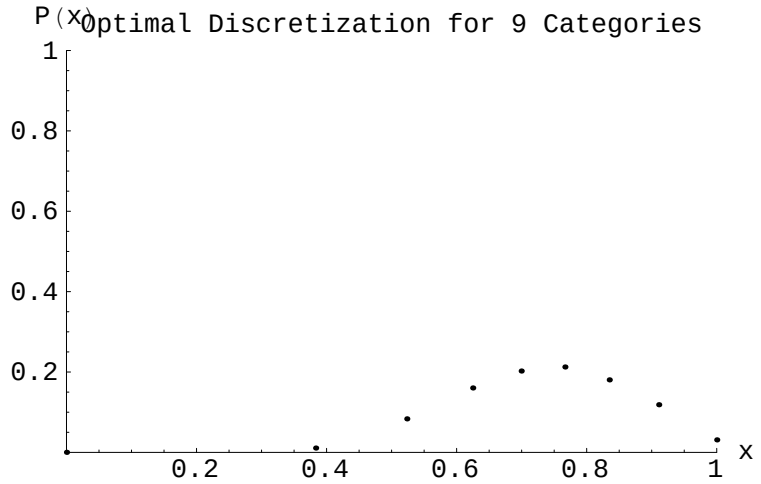

P(x)
Optimal Discretization for 4 Categories
1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1
x


P(x)
Optimal Discretization for 5 Categories
1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1
x


P(x)
Optimal Discretization for 6 Categories
1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1
x


P(x)
Optimal Discretization for 7 Categories
1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1
x


P(x)
Optimal Discretization for 8 Categories
1
0.8
0.6
0.4
0.2
0.2
0.4
0.6
0.8
1
x



■ Performing Scattered and Gridded Data Interpolations

The basic situation is that, for non-discrete models, the sets of boundary points which would completely define the system's structure function are uncountably infininite in number. Thus, we cannot reasonably expect the customer to specify the system completely using boundary points. Some authors have used this point to argue that the continuous model is infeasible.

However, there is another option: taking whatever data points the customer can give you, and using them to construct a multi-dimensional interpolation to the customer's true structure function. We consider this technique here.

First, let's consider a situation where the customer gives us only five data points (including the extrema):

```
inp = {{{0,0},0},{{0,1/2},1/2},{{1/2,0},1/2},
       {{1/2,1/2},3/4},{{1,1},1}};
```

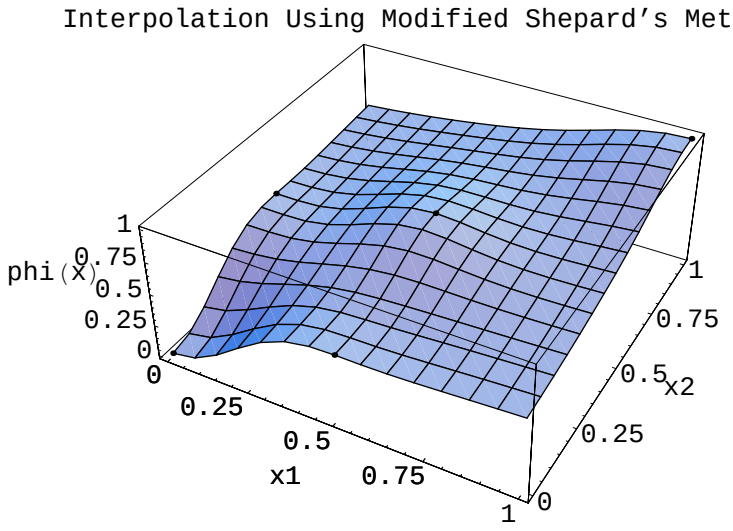
We first verify that the customer's input is coherent (note, though: coherence in this context only means "non-decreasing"- the customer has not after all specified a complete structure function):

```
CohInputQ[inp]
```

```
True
```

Now, we plot the Shepard interpolation to the structure function for this two-component system. Note that there are "bumps" near the data points that the customer supplied. This is a characteristic of Shepard's method (the points on the graphs are the customer supplied data points). This method is "modified" only in the sense that the values of the function at the customer supplied data points are equal to those points: in raw form, Shepard's method is not defined at those points, though it approaches them smoothly from all directions.

```
Show[Plot3D[Shepard[{x1,x2},inp],{x1,0,1},{x2,0,1},
            DisplayFunction->Identity],
     Graphics3D[Point /@ Flatten /@ inp,
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{"x1","x2","phi(x)"},
PlotLabel->
   "Interpolation Using Modified Shepard's Method"]
```




```
- Graphics3D -
```

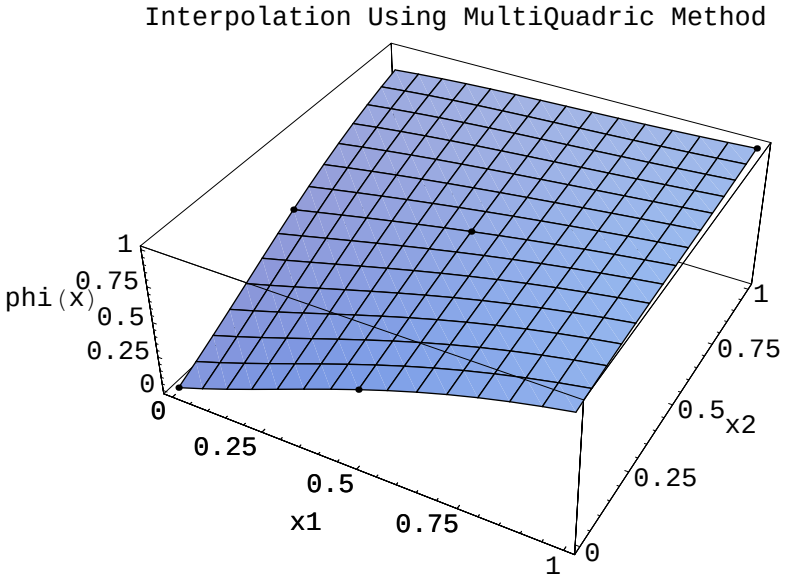
Now, we do the same interpolation using the MultiQuadric method. Note that the constant weights to the MultiQuadric method must first be calculated. Also, note that a value for RSQ of 1 will be used throughout this example.

```
inpc=MultiQuadricC[inp,1]
```

```
{7.20473, -5.10365, -5.10365, 2.71013, 0.512785}
```

Here is the plot of the MultiQuadric interpolation. Note that it is much smoother than the Shepard's method interpolation.

```
Show[Plot3D[MultiQuadric[{x1,x2},inp,inpc,1],
       {x1,0,1},{x2,0,1},
            DisplayFunction->Identity],
     Graphics3D[Point /@ Flatten /@ inp,
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{"x1","x2","phi(x)"},
     PlotLabel->"Interpolation Using MultiQuadric Method"]
```



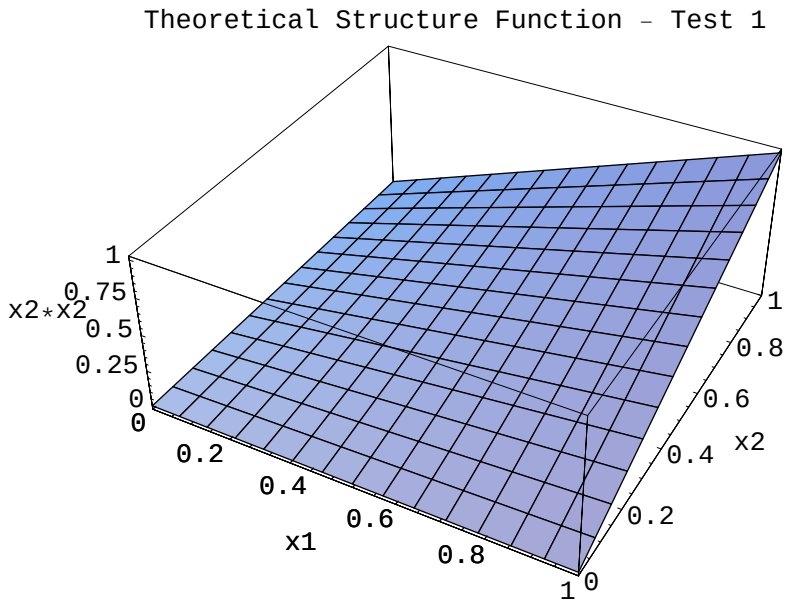

```
- Graphics3D -
```

Now, we examine situation where the customer data points are generated from some function, which for the purposes of analysis will from then on be treated as unknown. For the purposes of illustrating the differences between the Shepard'smethod and the MultiQuadric method, this is essential as it will allow us to compare the results of the two methods to a common reference point:

```
phi[x_] := Times @@ x
```

Here is a plot of this theoretical structure function:

```
Plot3D[phi[{x1,x2}],{x1,0,1},{x2,0,1},
  AxesLabel->{"x1","x2","x2*x2"},
  PlotLabel->"Theoretical Structure Function - Test 1"]
```

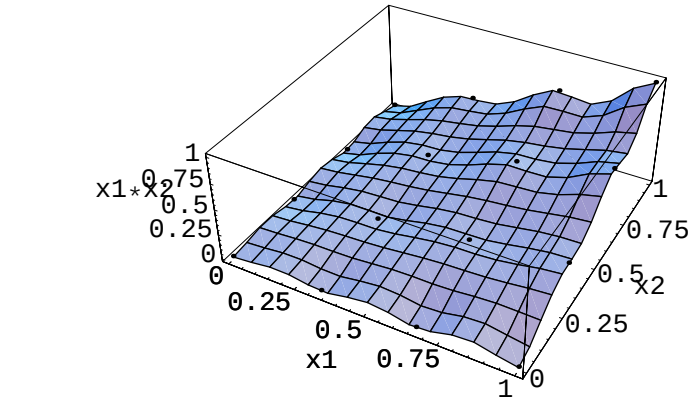



```
- SurfaceGraphics -
```

First, we interpolated based on gridded data (the normal case will be scattered data). We first use Shepard's method, then the MultiQuadric method.

```
test1=PhiGrid[phi];
```

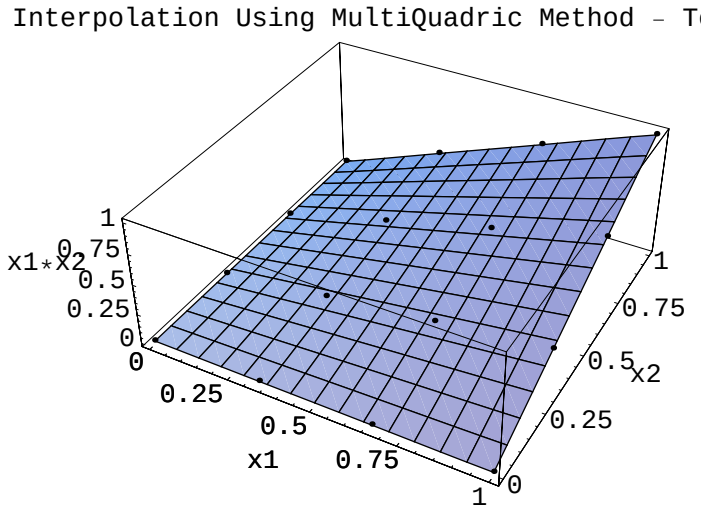
```
Show[Plot3D[Shepard[{x1,x2},test1],{x1,0,1},{x2,0,1},
            DisplayFunction->Identity],
     Graphics3D[Point /@ ((Flatten /@ test1)+
        Table[{0,0,0.01},{Length[test1]}]),
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{"x1","x2","x1*x2"},
PlotLabel->
"Interpolation Using Modified Shepard's Method - Test 1"]
```



- Graphics3D -

```
test1c=MultiQuadricC[test1,1];
```

```
Show[Plot3D[MultiQuadric[{x1,x2},test1,test1c,1],
       {x1,0,1},{x2,0,1},
            DisplayFunction->Identity],
     Graphics3D[Point /@ ((Flatten /@ test1)+
           Table[{0,0,0.01},{Length[test1]}]),
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{"x1","x2","x1*x2"},
PlotLabel->
"Interpolation Using MultiQuadric Method - Test 1"]
```

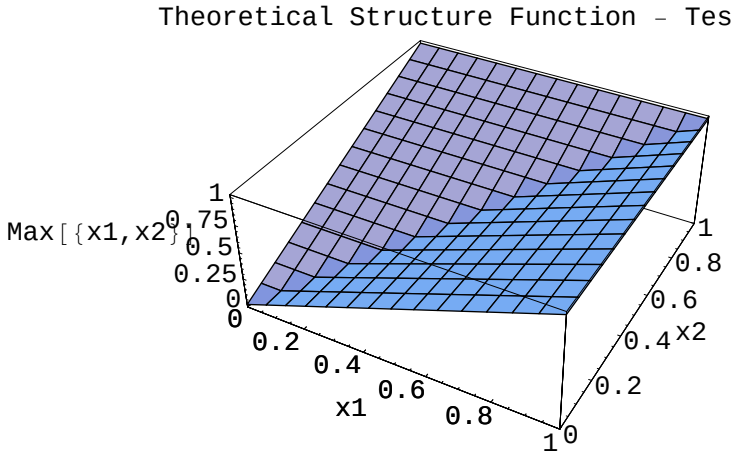



```
- Graphics3D -
```

Now, we perform the same analysis with yet another structure function. However, in this case the data will be random rather than gridded:

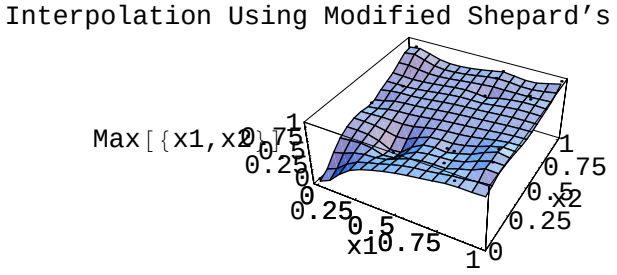
```
phi2[x_] := Max[x]

test2=ExtremaAdd[RandomPhi[phi2,16]];
```

```
Plot3D[phi2[{x1,x2}],{x1,0,1},{x2,0,1},
   AxesLabel->{"x1","x2","Max[{x1,x2}]"},
   PlotLabel->"Theoretical Structure Function - Test 2"]
```



- SurfaceGraphics -

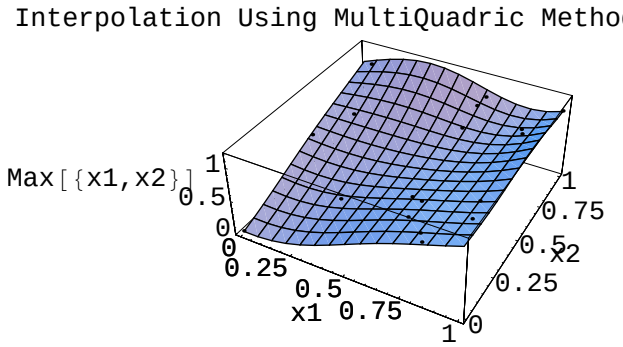
```
Show[Plot3D[Shepard[{x1,x2},test2],{x1,0,1},{x2,0,1},
            DisplayFunction->Identity],
     Graphics3D[Point /@ ((Flatten /@ test2)+
        Table[{0,0,0.05},{Length[test2]}]),
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{"x1","x2","Max[{x1,x2}]"},
PlotLabel->
"Interpolation Using Modified Shepard's Method - Test 2"]
```




```
- Graphics3D -
```

```
test2c=MultiQuadricC[test2,1];
```

```
Show[Plot3D[MultiQuadric[{x1,x2},test2,test2c,1],
    {x1,0,1},{x2,0,1},
            DisplayFunction->Identity],
     Graphics3D[Point /@ ((Flatten /@ test2)+
        Table[{0,0,0.01},{Length[test2]}]),
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{"x1","x2","Max[{x1,x2}]"},
PlotLabel->
   "Interpolation Using MultiQuadric Method - Test 2"]
```




```
- Graphics3D -
```

For this last example, we wish to quantify the difference between our interpolations and the real values of the function. This will be done by comparing the differences between our estimate of the structure function and the real values at a set of points on a regular grid. Note that your results may be different if you recalculate this notebook, due to the random nature of the data sets.

```
ShepardRMSError[test2,phi2]
```

```
0.112412
```

```
MultiQuadricRMSError[test2,phi2,10,1]

0.053017
```

There is one important point that so far has been missed in this analysis. We bothered to check the customer input in the first case to see that it was non-decreasing, and in subsequent cases generated the data from functions that we knew to be non-decreasing. However, are we assured that our interpolation is non-decreasing, especially when the input data is scattered? Special "ND" versions of these functions were written to alleviate concerns of loss of coherence. See the function usage statements (or the documentation earlier in this chapter) for details on how this is done).

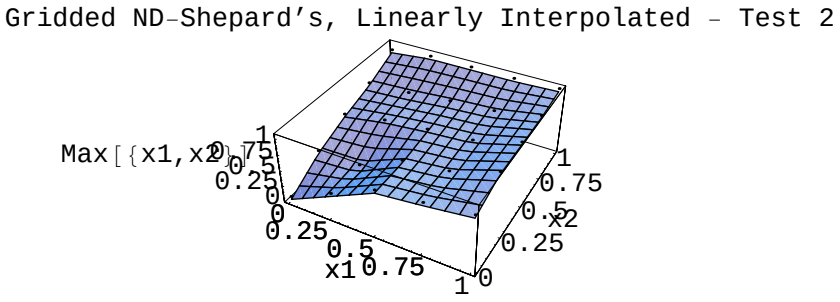
Unlike the regular non-"ND" functions, these ND functions return a regular grid of interpolated points, which can then be passed to MLinInt[] to linearly interpolate other points between the grid points. It is assumed that this linear interpolation is non-decreasing, if the underlying grid values are non-decreasing.

Frist, we note that ShepardND produces a non-decreasing grid, while Shepard does not, for the same data set (again, if you recalculate test2, this may not be the case for your particular randomization):

```
ans=ShepardND[test2];

CohInputQ[ans]

True

ans2=ShepardGrid[test2];

CohInputQ[ans2]

2, 6

4, 8

5, 6

9, 10

14, 15

False
```

Now, here is the full function estimate and graph for the Non-Decreasing Shepard's method:

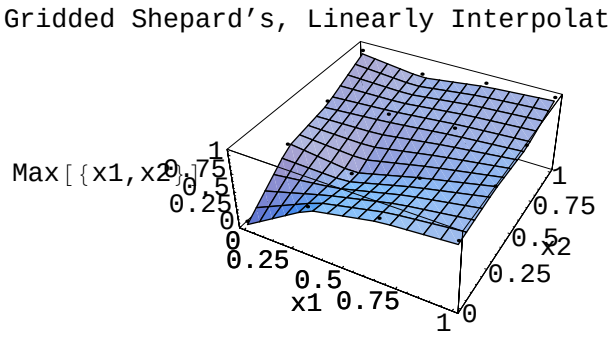
```
Show[Plot3D[MLinInt[{x1,x2},ans],{x1,0,1},{x2,0,1},
            DisplayFunction->Identity],
     Graphics3D[Point /@ ((Flatten /@ ans)+
        Table[{0,0,0.05},{Length[ans]}]),
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{"x1","x2","Max[{x1,x2}]"},
PlotLabel->
"Gridded ND-Shepard's, Linearly Interpolated - Test 2"]
```




```
- Graphics3D -
```

By way of comparison, here is the identical graph, but for the regular Shepard's method.

```
Show[Plot3D[MLinInt[{x1,x2},ans2],{x1,0,1},{x2,0,1},
            DisplayFunction->Identity],
     Graphics3D[Point /@ ((Flatten /@ ans2)+
        Table[{0,0,0.05},{Length[ans2]}]),
            DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{"x1","x2","Max[{x1,x2}]"},
PlotLabel->
"Gridded Shepard's, Linearly Interpolated - Test 2"]
```




```
- Graphics3D -
```

We note that the ND method actually has a better RMSError number for this example.

```
GriddedRMSError[ans,phi2]
```

```
0.0852042
```

```
GriddedRMSError[ans2,phi2]
```

```
0.11883
```

Now going back to our earlier multiplicativestructure function, we note that there is a vast differencebetweenthe performanceof the MultiQuadricmethod and the Shepard'smethod in this case.

```
inp2=PhiGrid[(Times @@ #)&];

ans2shep=ShepardND[inp2];

ans2mq=MultiQuadricND[inp2,5,1];
```

```
GriddedRMSError[ans2shep,(Times @@ #)&] // N
```

```
0.0602315
```

```
GriddedRMSError[ans2mq,(Times @@ #)&] // N
```

```
0.000528323
```

■ Determining Optimal RSQ Values

The question of what value to use for RSQ in the MultiQuadric interpolation is a matter of some theoretical debate. A few papers have been published containing general guidelines, but the fact remains that the value of RSQ can make a LARGE difference in the quality of fit one obtains, and that the problem is highly problem dependent. Thus, it was deemed of value to attempt to discern, for the type of problem the MultiQuadric method is being used for here, what values of RSQ might be reasonable.

First, we define a few structure functions that are commonly used in reliability studies:

```
phi1[x_] := Min[x]
```

```
phi2[x_] := Max[x]
```

Now, we calculate a random set of 20 data points in the (x1,x2) plane. These will be used in combination with the value of the (assumed) structure function at that point to create a data set. In general, our plan is to be able to compare a MultiQuadric interpolation (obtained from a set of n data points) for some curve to values of that curve at points other than the data points. If the average squared deviation of the curve fit from the real value at these test points is low, then we will say that the fit is good. We are looking for the lowest possible value of this mean root mean squared deviation, as a function of rsq. Are assumption will then be that since this is a typical sort of reliability problem with a typical number of customer-supplied data points, that our estimates of the optimal RSQ value should give us guidance as to what sort of default value to set for RSQ.

```
rawdata=RandomGenerate[20,2];
```

```
TableForm[rawdata, TableSpacing->{0}]
```

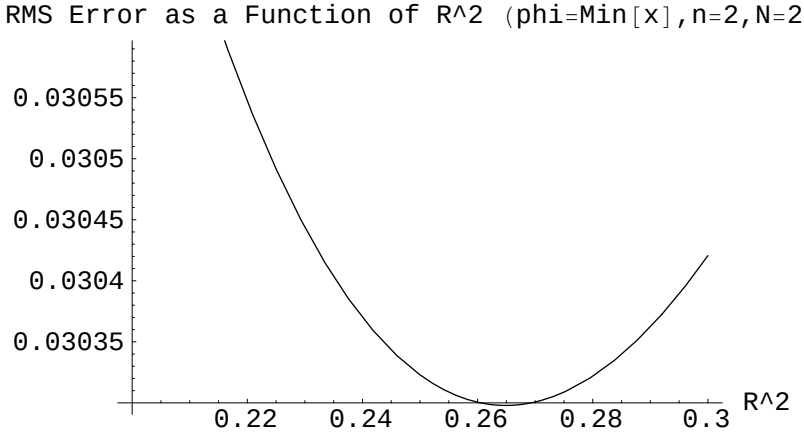
```
0.522751    0.198628
0.35554     0.559014
0.628485    0.397655
0.226962    0.873327
0.937107    0.236822
0.296094    0.584736
0.108947    0.22513
0.604372    0.313554
0.156146    0.0543806
0.968775    0.0148952
0.839124    0.790684
0.960192    0.491718
0.316374    0.592056
0.604652    0.932704
0.687888    0.194401
0.37769     0.0593762
0.750782    0.957579
0.0815961   0.47464
0.641834    0.732449
0.477224    0.161086
```

Now, as an assumption of coherent systems is that the maximal state vector maps into the maximal system state, and that the minimal state vector maps into the minimal system state, we add this additional piece of knowledge. We now have a total of 22 data points.

```
data1=ExtremaAdd[{#, phi1[#]}& /@ rawdata];

data2=ExtremaAdd[{#, phi2[#]}& /@ rawdata];
```

Now, we examine a graph of the error as a function of RSQ.

```
Plot[MultiQuadricRMSError[data1,phi1,5,rsq],
  {rsq,0.2,0.3},
 PlotLabel->
  "RMS Error as a Function of R^2 (phi=Min[x],n=2,N=22)",
 AxesLabel->{"R^2",""}]
```




```
- Graphics -
```

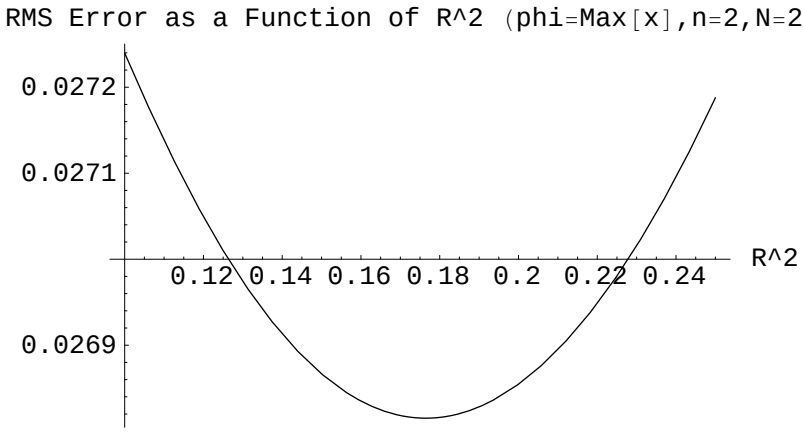
Now, optimizing, we find the optimal RSQ value for this problem

```
FindMinimum[MultiQuadricRMSError[data1,phi1,5,rsq],
   {rsq,0.075,0.080}]
```

```
{0.0302979, {rsq → 0.264888}}
```

We proceed with the same analysis, using a different structure function.

```
Plot[MultiQuadricRMSError[data2,phi2,5,rsq],
 {rsq,0.10,0.25},
PlotLabel->
"RMS Error as a Function of R^2 (phi=Max[x],n=2,N=22)",
AxesLabel->{"R^2",""}]
```




```
- Graphics -
```

```
FindMinimum[MultiQuadricRMSError[data2,phi2,5,rsq],
   {rsq,0.040,0.038}]
```

```
{0.0268152 , {rsq → 0.176478 }}
```

Now, for the first structure function, we perform the optimization 15 different times on 15 different data sets, to find the mean and variance of the optimal RSQ value obtained.

```
b=Table[RandomGenerate[20,2],{15}];

ansb=Table[rsq /. FindMinimum[MultiQuadricRMSError[
      ExtremaAdd[{#, phi1[#]}& /@ b[[i]]],
      phi1,5,rsq],{rsq,0.075,0.080}][[2]],
         {i,Length[b]}];
```

```
TableForm[ansb,TableSpacing->{0}]
```

```
0.102249
0.08042
0.110037
0.196671
0.14035
0.402902
0.0834257
0.0740417
0.15372
0.207568
0.135696
0.180243
0.0371163
0.25137
0.259234
```

We now request the mean and standard deviation of the set of optimal RSQ values obtained. The only thing that was different between these data sets was the placement of 20 of the 22 data points.

```
Apply[Plus, ansb] / Length[ansb]
```

```
0.161003
```

```
Sqrt[Apply[Plus, (ansb - Apply[Plus, ansb] /
     Length[ansb])^2]/(Length[ansb] - 1)]
```

```
0.0936303
```

Now, we wish to examine what the optimal value of RSQ is for systems of other sizes, and for data sets of different sizes.

```
MaxROpt[b_] := FindMinimum[MultiQuadricRMSError[
      ExtremaAdd[{#, phi1[#]}& /@ b],
      phi1,5,rsq],{rsq,0.075,0.080}]
```

Here is a system with two components, with 100 customer-supplieddata points.

```
b1=RandomGenerate[100,2];
```

```
MaxROpt[b1]
```

```
{0.0146579 , {rsq → 0.074963 }}
```

Here is a system with two components, with 50 customer-supplieddata points.

```
b2=RandomGenerate[50,2];
```

```
MaxROpt[b2]
```

```
{0.0161703 , {rsq → 0.0833332 }}
```

Here is another system with two components, with 50 customer-supplieddata points.

```
b4=RandomGenerate[50,2];
```

```
MaxROpt[b4]
```

```
{0.0105009 , {rsq → 0.0480841 }}
```

Here is a system with two components, with 20 customer-supplieddata points.

```
b3=RandomGenerate[20,2];
```

```
MaxROpt[b3]
```

```
{0.0474718 , {rsq → 0.168456 }}
```

Here is a system with three components, with 10 customer-supplieddata points.

```
b5=RandomGenerate[10,3];
```

```
MaxROpt[b5]
```

```
{0.150134 , {rsq → 0.28062 }}
```

The difficulty, of course, is that in real life one would not know the "real" structure function, and so one would have no way of knowing how close one'sinterpolationis to points not in the given data set (and so, one would have no way of knowing how good one'schoice of RSQ is). All that can be done is to choose a value that seems reasonable given past history with known problems one can assume to be of similar type.

We note that no optimal value of RSQ found was greater than 0.5. A default value of 1/6 was set for the MultiQuadric function in this package. However, it may be changed for other problems, if a value other than the default one seems appropriate.

The NewFunctions Package

As many of the functions in this package are featured and illustrated in the example packages accompanying these sections, for some only their "usage" statements will be presented here.

■ Function Documentation

■ Bounds2[data,f,ag:6]

This function finds bounds for reliability for continuous systems, based on the "maximal" and "minimal" structure functions. These are the structure functions consisting of the maximal and minimal points (respectively) that the given data points for each system allows, under the assumption of monotonicity. It will work for any number of components, but for large data sets may take a long period of time. The density function list f consists of the PDFs for the state of each component, where each is a function of x. Data is in the form that data is normally provided in for the MultiQuadric functions, etc. Ag is the AccuracyGoal of the Numerical Integration that this function contains. It has a maximal value of 6 - lower values will produce faster results. This function has been tested and found to work properly for systems with two and four components; it has not been tested for any systems with more components than four. Components are assumed to be independent, although this assumption could be easily relaxed if it may not be made for the problem at hand. If the list of density functions is indeed a list of CDF'sinstead, then ag may be set to 0 to tell Bounds2 that CDFs are provided; this results in faster computation, as numerical intergration need not be performed.

■ MultiQuadricND2[inpdata,rsq,prec]

This function performs the same function as MultiquadricND, but uses a newer and more efficient algorithm.

■ MultiQuadricNDRMSError[data,phi,m]

This function performs the same function as MultiquadricRMSError, but does so on the non-decreasing grid generated by MultiQuadricND or (preferably) MultiQuadricND2.

■ SystemFromDirectEnumerationLow[p,data,fphi,pprob]

This function will return the minimum system state probabilities of a discrete system, using the partial customer-supplied data set named data. This function calls systable2low.

■ SystemFromDirectEnumerationHigh[p,data,fphi,pprob]

This function will return the maximum system state probabilities of a discrete system, using the partial customer-supplied data set named data. This function calls systable2high.

■ PhiMinRMSError[data,phi,m,perc]

This function allows calculation of the RMS error for a min structure function specified by data, given knowledge of phi, with grid density m.

■ PhiMaxRMSError[data,phi,m,perc]

This function allows calculation of the RMS error for a max structure function specified by data, given knowledge of phi, with grid density m.

■ phiround[x,phi,fphi]

This function will return a value representing the element of the set fphi which is closest to phi[x].

■ MuSigma[inp,expr]

This function will find an estimate of the mean and standard deviation of a system, given this information for the components and a structure function, by using multi-dimensional Taylor expansion. Specifically, it returns a list of two elements, where the first is the estimated mean of the system in question, and the second is the estimated standard deviation of the system in question. The input is in the form [{var1, muvar1, sigmavar1}, {var1, muvar1, sigmavar1}, ...}, and expr is a function of each of the vari variables.

Here is an example:

```
params = MuSigma[{{r1, 3 / 4, 1 / 10},
                  {r2, 2 / 3, 1 / 10}}, (r1 + r2^3) / 2]
```

```
{0.523148 , 0.0833333 }
```

■ SVStillOut[mmax]

For Erlangian multistate components with m+1 states (beginning in their maximal states at t=0), where each state and the probability of that state are given by {i/m,p(i,t)}, 0<=i<=m, one may calculate the point mu*t where the state variance reaches its maximum, so that for a given value of mu one may know the specific time at which the component is at its maximum variance. This function will return a table of these values for given values of mup to the given value of mmax, for different sizes of systems.

Here is an example, which returns the values of mu*t for given values of m up to m=10:

```
SVStillOut[10] // TableForm
```

```
1       0.693147
2       1.22042
3       1.79764
4       2.40894
5       3.04585
6       3.70298
7       4.37674
8       5.06452
9       5.76436
10      6.47473
```

Binary DeterministicAnalysis Tutorial

■ Introduction

This document is a brief, basic tutorial on the application of some of the functions in the DeterministicAnalysis package to binary models.

■ Basic Functions

■ Defining the System

Let us define a simple binary system with four components. Note that we may name the variables anything we wish, and still pass them properly to the appropriate functions. They are given their traditional names here for clarity.

First, we define the component state space:

```
p = {{0,1},{0,1},{0,1},{0,1}}
```

```
{{0, 1}, {0, 1}, {0, 1}, {0, 1}}
```

Each element of p is the set of possible states for a given component of the system. Since this is a binary model, there are only two possible states for each of the four components of the system.

Now, we must define the system state space:

```
fphi = {0,1}
```

```
{0, 1}
```

Since this is a binary model, there are only two possible states of functioning for the system.

Of course, we may use any of the pre-existing functions which are part of every *Mathematica* environment. Let us assume that we wish to know how many components comprise our system. We could use the following command:

```
Length[p]
```

```
4
```

We may of course review the set of possible states for the components by requesting that p be printed again (or we could just scroll up the screen to where it was initially printed):

```
p
```

```
{{0, 1}, {0, 1}, {0, 1}, {0, 1}}
```

Now we should define the structure function which maps the states of the components into the states of the system. Our function phi should accept any combination of elements in p and return some element in fphi. We may use any *Mathematica* function we wish to define the structure function (there are literally hundreds), and we may individually define different values of fphi for different component state vectors. If we wished to, we could specify the entire system by direct enumeration. However, it is hoped for simplicity'ssake that one function or a small set of functions will perform this mapping for us.

Let us assume that we wish to examine a system where the system state is the minimum of all the component states. As there are no states possible for any of the components which is not also possible for the system to assume, this structure function is well-defined.

```
phi[x_] := Min[x]
```

Now we have defined a *Mathematica* function of our own, which we will use in subsequent calculations. Of course, when we call this function, we will be calling it with a parameter x. This parameter x represents a component state vector and is a list of real numbers. Note that the underscore after the x and the colon before the = sign in the above definition are both necessary.

Now that we have defined the set of values each component may assume (p), the set of values which the system can assume (fphi), and the relationship between every possible component vector and the system state (phi), we have completely specificed the structural behavior of the model. We may now begin to make calculations.

■ Determining System Coherence

The first thing we should do with any system is to determine whether it is coherent or not. If a system is coherent, we may bring to bear on the problem all of the theorms that have been proved for coherent systems. Of particular importance is to know whether the system is non-decreasing; if it is, we may speak of boundary points in a mathematically well-defined way. The following example shows how to check that a system is non-decreasing:

```
NonDecreasingQ[p,phi]
```

```
True
```

The system is non-decreasing. Now, we can check the limits of the system, to be sure that the maximal component state vector is mapped into the maximal system state, and that the minimal component state vector is mapped into the minimal system state. Systems that fail this test and which are non-decreasing are indicative of a poorly defined system state space that includes maximal states which may not be achieved even when every component functions perfectly.

```
ProperLimitsQ[p,phi,fphi]
```

```
True
```

Thus, the system maps the maximal and minimal component vectors properly. Now, we check the component relevence. If a component is irrelevent, then there is no component state vector where the performance of that particular component will affect the system state. Systems that have irrelevent components are indicative of a poorly defined system where the model is unnecessarily large.

```
ReleventComponentsQ[p,phi]
```

```
True
```

Thus, since the system passes all three of these relevence tests, it is coherent. These tests (along with every other function in the DeterministicAnalysis package) work equally as well with multistate models (with non-identical finite numbers of real states for every component and for the system) as they do with binary models.

Of course, since we will wish to use this test on many different systems, and since a system must pass all three tests to be coherent, we may wish to use one function, to check all three at once. If it returns true, the system is coherent.

```
CoherentQ[p,phi,fphi]
```

```
True
```

■ Discerning and Manipulating the Boundary Points

Now, let us proceed with our analysis by finding and manipulating the upper and lower boundary points to this system. Note that, for strictly binary models, there is a simplified form in which the paths and cuts may be written; see the section labelled "Binary Model Simplification Functions" for more information. However, as our goal here is to understand how to use these functions for general multistate models, the full form of upper and lower boundary points will be used for the remainder of this particular section.

We may find the lower boundary points with the following function:

```
lbps=LBPFromStructure[p,phi,fphi]
```

```
{{{1, 1, 1, 1}, 1, Lower, Real}}
```

And we may find the upper boundary points with the following function:

```
ubps=UBPFromStructure[p,phi,fphi]
```

```
{{{0, 1, 1, 1}, 0, Upper, Real}, {{1, 0, 1, 1}, 0, Upper, Real},
 {{1, 1, 0, 1}, 0, Upper, Real}, {{1, 1, 1, 0}, 0, Upper, Real}}
```

Note the structure of the information which these functions have returned to us. ubps, for example, is a list containing four elements. Each element is an upper boundary point, indicating that this system has four upper boundary points. Each boundary point it itself a list, consisting of four elements. The first is the vector of states representing the boundary point. The second is the level that vector is a boundary point to. The third is either the string "Upper" or the string "Lower" to tag which type of boundary point the given one is. The fourth and final element in the list representing each element is a string, which is either "Real", "Virtual", or "Indet."

Strictly speaking, this last element is not necessary, but it is sometimes interesting. A boundary point is termed "Real" if the value of the system is precisely that given level for the given boundary point. It is termed "Virtual" if it is either above that value (for lower boundary points) or below that value (for upper boundary points). This situation normally occurs when there are system states which no component state vector maps to, and sometimes in other cases too. The string in the fourth position is "Indet" if it is not known whether the boundary point is "Real" or "Virtual". In any case, the forms of ubps and lbps above are example of the form the set of lower and upper boundary points must take when passed to functions in this package that require them.

Let us say that we were given only the upper boundary points, or perhaps the lower boundary points. It would be quite unnecessary for the customer to specify both, when one set can be determined from the other. If we were given the lower boundary points, we could obtain the upper ones with the following function:

```
LBPToUBP[lbps,p,fphi]
```

```
{{{0, 1, 1, 1}, 0, Upper, Indet}, {{1, 0, 1, 1}, 0, Upper, Indet},
 {{1, 1, 0, 1}, 0, Upper, Indet}, {{1, 1, 1, 0}, 0, Upper, Indet}}
```

Note that this list of boundary points is the same as that which was obtained earlier. The only difference is that "Indet" is used in place of "Real." This is because Real/Virtual/Indet calculations are made only in LBPFromStructure[] and UBPFromStructure[] as a matter of curiosity: they are never used computationally (though it would be most accurate to say that the *designation* is not used computationally- a system with virtual boundary points that are removed will be an ENTIRELY different system).

One may of course also calcuate the lower boundary points based on the upper boundary points:

```
UBPToLBP[ubps,p,fphi]
```

```
{{{1, 1, 1, 1}, 1, Lower , Indet }}
```

Of course, it is often true that we may not have the structure function for a given problem: just the upper or lower boundary points as specified by the customer. Even if we assume that we don't have the structure function, the same range of analysis is still available to us.

We can establish the relevence of all the components with only the upper and lower boundary points using the function BoedigheimerReleventComponentsQ[]:

```
BoedigheimerReleventComponentsQ[lbps,ubps,p]
```

```
True
```

Let us say that we doubt the self-consistency of the boundary points we have been given, and wish to check them. The following two functions will check the self consistency of each of the two types of boundary points:

```
LBPSelfConsistentQ[lbps]
```

```
True
```

```
UBPSelfConsistentQ[ubps]
```

```
True
```

We can also check the boundary point sets for consistency with each other, using the following function:

```
BPConsistentToEachOtherQ[lbps,ubps]
```

```
True
```

Boedigheimer[1992] created definitions for series and parallel that rely only on knowledge of the upper and lower boundary points. Two functions will check if (using this definition) the system is parallel or series based on the two sets of boundary points.

```
BoedigheimerParallelQ[lbps,ubps,fphi]
```

```
False
```

```
BoedigheimerSeriesQ[lbps,ubps,fphi]
```

```
True
```

Note that in the binary case series systems are defined as ones where the structure function is the minimum of all of the components, or (equivalently) that the structure function is the product of all of the system states. We would expect the system to be series, then, and as we can see above Boedigheimer's definition of series is consistent in this case with the usual definition.

■ Discerning Structure Function Based on Boundary Points

One can discern the system state associated with any component vector based on the boundary points by using following two functions:

```
SystemStateFromLBP[lbps,fphi,{1,1,0,1}]
```

```
0
```

```
SystemStateFromLBP[lbps,fphi,{1,1,1,1}]
```

```
1
```

■ Binary Model Simplification Functions

If you are accustomed to working with binary models exclusively, the full form of the lists of upper and lower boundary points may seem awkward to you. For strictly binary models, we may use some special functions to convert the general list of boundary points to the traditional form of paths and cuts:

```
PathsFromLBP[lbps]
```

```
{{1, 2, 3, 4}}
```

```
CutsFromUBP[ubps]
```

```
{{1}, {2}, {3}, {4}}
```

We may also specify the lower and upper boundary points for binary models in this simplified form, if we wish, and then convert them to full form for further processing:

```
LBPFromPaths[{{1,2,3,4}},4]
```

```
{{{1, 1, 1, 1}, 1, Lower, Real}}
```

```
UBPFromCuts[{{1},{2},{3},{4}},4]
```

```
{{{0, 1, 1, 1}, 0, Upper, Real}, {{1, 0, 1, 1}, 0, Upper, Real},
 {{1, 1, 0, 1}, 0, Upper, Real}, {{1, 1, 1, 0}, 0, Upper, Real}}
```

As an example of how Mathematica functions may be nested, consider the PathsToCuts[] function, which is defined as follows:

```
??PathsToCuts
```

```
PathsToCuts[paths,n] returns the minimal cuts based on the
  minimals paths for a system of order n.

Attributes[PathsToCuts] = {Protected}

PathsToCuts[paths_List, n_Integer] := CutsFromUBP[
     LBPToUBP[LBPFromPaths[paths, n], Table[{0, 1}, {n}], {0, 1}]]

PathsToCuts[___] :=
   Message[DeterministicAnalysisFunction::badcall, "PathsToCuts"]
```

As you can see, it accepts as the innermost argument the simplified binary form of path specification, converts it to full form, passes that full form to the LBPToUBP[] function for conversion, and then converts the full-form results to the simplified binary form of cut specification. Here is an example:

```
PathsToCuts[{{1,2,3,4}},4]
```

```
{{1}, {2}, {3}, {4}}
```

```
CutsToPaths[{{1},{2},{3},{4}},4]
```

```
{{1, 2, 3, 4}}
```

■ Vector Comparison

LessOrEqualQ[x,y] discerns whether the vector x is less than or equal to the vector y, in the sense that is usually meant in the study of multistate reliability.

```
LessOrEqualQ[{1,0,1},{1,1,1}]
```

```
True
```

There are similar functions LessQ[x,y], GreaterOrEqual[x,y], GreaterQ[x,y].

■ Component State Combinations: VectorSpace[p]

VectorSpace[p] takes a system state space variable and returns a list of all the possible combinations of states. This may be useful if you want to create a map of all the possible component vectors to their associated system states.

```
VectorSpace[p]
```

```
{{0, 0, 0, 0}, {0, 0, 0, 1}, {0, 0, 1, 0}, {0, 0, 1, 1},
 {0, 1, 0, 0}, {0, 1, 0, 1}, {0, 1, 1, 0}, {0, 1, 1, 1},
 {1, 0, 0, 0}, {1, 0, 0, 1}, {1, 0, 1, 0}, {1, 0, 1, 1},
 {1, 1, 0, 0}, {1, 1, 0, 1}, {1, 1, 1, 0}, {1, 1, 1, 1}}
```

We could use our knowledge of Mathematica to create a list, where each component contains both the component vector and the system state associated with it, for all possible component vectors.

```
{#, phi[#]}& /@ VectorSpace[p]
```

```
{{{0, 0, 0, 0}, 0}, {{0, 0, 0, 1}, 0}, {{0, 0, 1, 0}, 0},
 {{0, 0, 1, 1}, 0}, {{0, 1, 0, 0}, 0}, {{0, 1, 0, 1}, 0},
 {{0, 1, 1, 0}, 0}, {{0, 1, 1, 1}, 0}, {{1, 0, 0, 0}, 0},
 {{1, 0, 0, 1}, 0}, {{1, 0, 1, 0}, 0}, {{1, 0, 1, 1}, 0},
 {{1, 1, 0, 0}, 0}, {{1, 1, 0, 1}, 0}, {{1, 1, 1, 0}, 0},
 {{1, 1, 1, 1}, 1}}
```

■ Structural Importance: StructuralImportances[p,phi]

The function StructuralImportances[p,phi] is available to discern structural importances. It returns a vector of length n consisting of the structural importances of each component, and is based on the general definition given on Boedigheimer[1992]. Here is an example of its use:

```
StructuralImportances[p,phi]
```

$$\left\{\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}\right\}$$

Note that an irrelevent component has a structural importance of zero, as illustrated below:

```
phi2[x_] := Min[x[[1]],x[[2]],x[[3]]]
```

```
StructuralImportances[p,phi2]
```

$$\left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, 0\right\}$$

Of course, we would ordinarily have noticed this possibility by utilizing either of the following two functions:

```
ReleventComponentsQ[p,phi2]
```

```
False
```

```
CoherentQ[p,phi2,fphi]
```

```
False
```

■ Limit on Number of Critical Upper Vectors - CUVUpperBound[n,m]

In Xue and Yang ("Symmetric Relations in Multistate Systems", 1995), a method is outlined to discover an upper limit on the number of "critical upper vectors," which are related to the boundary points we have been utilizing in this document. This function finds the upper limit on the number of critical upper vectors, given that the system has n components and each component and the system has exactly m+1 states. Here is an example:

```
CUVUpperBound[7,1]
```

```
35
```

```
CUVUpperBound[7,2]
```

```
393
```

```
CUVUpperBound[7,7]
```

```
134512
```

As one might suspect, the complexity of a given multistate system increases rapidly as the number of states per component increases.

■ Boundary Point Set Completion - BPClean[lbps or ubps]

Another function, BPClean[], is intended to make data entry for lower and upper boundary point sets easier. Although a few extra features are built into this function that allow you to specify in a shorthand way the types of upper or lower boundary point (see function documentation for details), at a basic level this function will allow you to skip the last two string entries for each boundary point. This function will also sort the boundary point set properly, in ascending order of the level for each point. Here is an example:

```
lbpsshort={{{3,3,3},3},{{1,1,1},1},{{2,2,2},2}}
```

```
{{{3, 3, 3}, 3}, {{1, 1, 1}, 1}, {{2, 2, 2}, 2}}
```

```
BPClean[lbpsshort,"Lower","Real"]
```

```
{{{1, 1, 1}, 1, Lower, Real}, {{2, 2, 2}, 2, Lower, Real},
 {{3, 3, 3}, 3, Lower, Real}}
```

■ Strategies When the Structure Function is Not Known

Note that, with many systems, we do not have the explicit structure function for the system. Instead, we have the assumption that the structure function is non-decreasing and we have a set of either the upper or lower boundary points (commonly called minimal cuts and minimal paths in the binary case) which are provided by the customer. If this is the case, we may (first using LBPToUBP[] or UBPToLBP[] to get the other set of boundary points, if only one set is given, and then perhaps using UBPSelfConsistentQ[],LBPSelfConsistentQ[],and BPConsistentToEachOtherQ[] to check the internal self-consistency of the boundary points if we have doubts) use the function BoedigheimerReleventComponentsQ[] to check for component relevence based on the boundary points rather than the structure function.

If we can make the assumption that the set of boundary points we have been given is complete and properly sorted, we can examine the set of boundary points to see whether the system properly maps the minimal and maximal states. If the maximal state for which there is lower boundary point is equal to the maximal state in the set of states we are given for the system, and if the minimal state for which there is an upper boundary point is equal to the minimal state in the set of states we are given for the system, then ProperLimitsQ[] would have been satisfied.

Of course, another option is to use the functions SystemStateFromLBP[] or SystemStateFromUBP[] as part of the definition for phi[] (as they will return the system state of any given component state based on either the lower or upper boundary points) and then utilize CoherentQ[] as it is.

General Discrete Analysis Tutorial

■ Introduction

The goal of this document is to illustrate how one may use various RelPack packages to perform complex analyses quickly and easily on a sample problem. This tutorial uses only a fraction of all the functions which are available in RelPack 2.0, and does not utilize any of the more advanced packages (such as ContinuousOptimization).

We will begin by defining the system and graphing its behavior. Then we will, in turn, examine the deterministic, stochastic, and dynamic properties (both actual and potential) that arise from it.

■ System Definition

First, we must define the structure function, the component state vector space, and the system state space.

The *Mathematica* function below defines the system's structure function.

```
phi[x_] := Module[{systab},
   systab = {{{0,0},0},
             {{1,0},0},
             {{2,0},1},
             {{3,0},1},
             {{4,0},1},
             {{5,0},1},
             {{0,1},0},
             {{1,1},0},
             {{2,1},2},
             {{3,1},2},
             {{4,1},2},
             {{5,1},2},
             {{0,2},0},
             {{1,2},3},
             {{2,2},3},
             {{3,2},3},
             {{4,2},3},
             {{5,2},3},
             {{0,3},0},
             {{1,3},3},
             {{2,3},4},
             {{3,3},4},
             {{4,3},4},
             {{5,3},5},
             {{0,4},0},
             {{1,4},3},
             {{2,4},4},
```

Next, we define the state space of the system:

```
fphi = Range[0,5]
```

```
{0, 1, 2, 3, 4, 5}
```

Now, we define the state space of the component vectors:

```
p = {Range[0,5], Range[0,4]}
```

```
{{0, 1, 2, 3, 4, 5}, {0, 1, 2, 3, 4}}
```

To check our work, we request a table of the values of the structure function. x1 is along the horizontal axis, and x2 is along the vertical axis.

```
phitab = Table[phi[{x1,x2}], {x2, 4, 0, -1},
   {x1, 0, 5}];
```

```
MatrixForm[phitab]
```

$$\begin{pmatrix} 0 & 3 & 4 & 4 & 5 & 5 \\ 0 & 3 & 4 & 4 & 4 & 5 \\ 0 & 3 & 3 & 3 & 3 & 3 \\ 0 & 0 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

■ Graphs of the Structure Function

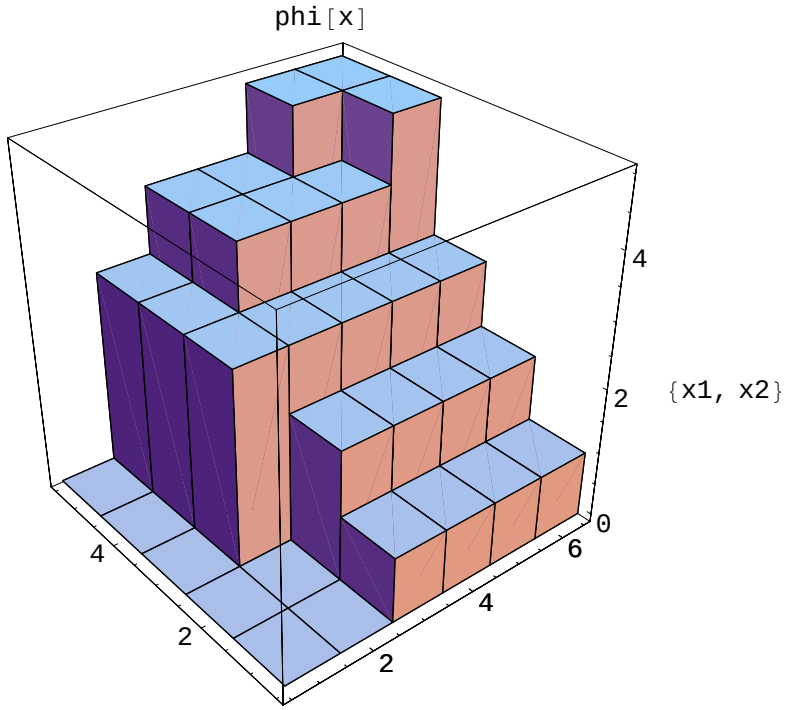
For ease of visualization, it might be wise to show a graph of the original structure function. First, we load a few standard Mathematica packages which assist in graphical work:

```
<<Graphics‘Graphics3D‘
```

```
<<Graphics‘Legend‘
```

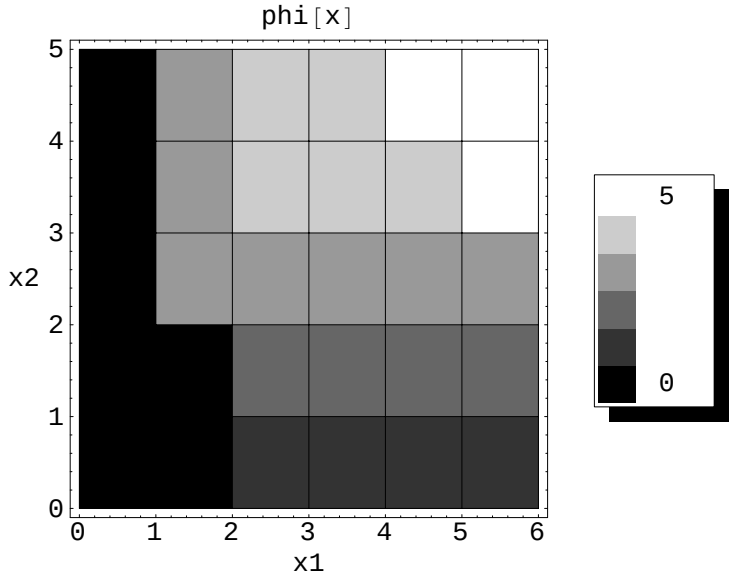
With these packages in place, we can view the system as a 3D Bar Chart:

```
BarChart3D[Table[phi[{x1,x2}],
      {x1, 0, 5}, {x2, 0, 4}],
   PlotLabel -> "phi[x]",
   AxesLabel -> {"x1","x2"},
   ViewPoint -> {-1.819, -2.233, 1.775}];
```



We can also view the system as a density plot:

```
ShowLegend[
   ListDensityPlot[Table[phi[{x1,x2}],
         {x2, 0, 4}, {x1, 0, 5}],
      PlotLabel -> "phi[x]",
      FrameLabel -> {x1,x2},
      RotateLabel -> False,
      DisplayFunction -> Identity],
   {GrayLevel[1-#]&, 6, " 5", " 0",
      LegendPosition -> {1.1, -0.4}}];
```



■ Deterministic Behavior of the System

First, we must verify that the system is coherent:

```
CoherentQ[p,phi,fphi]
```

```
True
```

Now, we can request the lower and upper boundary points:

```
lbps=LBPFromStructure[p,phi,fphi];
```

```
MatrixForm[lbps]
```

{1, 2}	1	Lower	Virtual
{2, 0}	1	Lower	Real
{1, 2}	2	Lower	Virtual
{2, 1}	2	Lower	Real
{1, 2}	3	Lower	Real
{2, 3}	4	Lower	Real
{4, 4}	5	Lower	Real
{5, 3}	5	Lower	Real

```
ubps=UBPFromStructure[p,phi,fphi];

MatrixForm[ubps]
```

{0, 4}	0	Upper	Real
{1, 1}	0	Upper	Real
{0, 4}	1	Upper	Virtual
{1, 1}	1	Upper	Virtual
{5, 0}	1	Upper	Real
{0, 4}	2	Upper	Virtual
{5, 1}	2	Upper	Real
{1, 4}	3	Upper	Real
{5, 2}	3	Upper	Real
{3, 4}	4	Upper	Real
{4, 3}	4	Upper	Real
{5, 2}	4	Upper	Virtual

Now, we check to see if the system states obtained using JUST the upper or lower boundary points are the same as those obtained when the whole structure function phi was used.

```
lbptab = Table[SystemStateFromLBP[lbps,fphi,{x1,x2}],
  {x2, 4, 0, -1}, {x1, 0, 5}];

MatrixForm[lbptab]
```

0	3	4	4	5	5
0	3	4	4	4	5
0	3	3	3	3	3
0	0	2	2	2	2
0	0	1	1	1	1

```
ubptab = Table[SystemStateFromUBP[ubps,fphi,{x1,x2}],
  {x2, 4, 0, -1}, {x1, 0, 5}];
```

```
MatrixForm[ubptab]
```

$$\begin{pmatrix} 0 & 3 & 4 & 4 & 5 & 5 \\ 0 & 3 & 4 & 4 & 4 & 5 \\ 0 & 3 & 3 & 3 & 3 & 3 \\ 0 & 0 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

As we can see and explicitly verify, the tables of values produced are identical.

```
phitab==lbptab==ubptab
```

```
True
```

Of course, many systems will be specified by the customer not through a structure function, but through the set of either the lower or the upper boundary points. For systems which are specified in this way, a function in this package called BPClean[] can significantly ease data entry. Also, there are a number of functions in this package which drastically simplify the data entry requirements when the system is strictly binary.

One question that has come up is whether the table of system state values produced by the upper/lower boundary point approach would be the same if the "virtual" boundary points were not included. Let us remove them and see...

```
(lbpsreal = Select[lbps,
   #[[4]]=="Real" &]) // MatrixForm
```

$$\begin{pmatrix} \{2, 0\} & 1 & \text{Lower} & \text{Real} \\ \{2, 1\} & 2 & \text{Lower} & \text{Real} \\ \{1, 2\} & 3 & \text{Lower} & \text{Real} \\ \{2, 3\} & 4 & \text{Lower} & \text{Real} \\ \{4, 4\} & 5 & \text{Lower} & \text{Real} \\ \{5, 3\} & 5 & \text{Lower} & \text{Real} \end{pmatrix}$$

```
(ubpsreal = Select[ubps,
   #[[4]]=="Real" &]) // MatrixForm
```

$$\begin{pmatrix} \{0, 4\} & 0 & \text{Upper} & \text{Real} \\ \{1, 1\} & 0 & \text{Upper} & \text{Real} \\ \{5, 0\} & 1 & \text{Upper} & \text{Real} \\ \{5, 1\} & 2 & \text{Upper} & \text{Real} \\ \{1, 4\} & 3 & \text{Upper} & \text{Real} \\ \{5, 2\} & 3 & \text{Upper} & \text{Real} \\ \{3, 4\} & 4 & \text{Upper} & \text{Real} \\ \{4, 3\} & 4 & \text{Upper} & \text{Real} \end{pmatrix}$$

```
(lbptabr =
  Table[SystemStateFromLBP[lbpsreal,fphi,{x1,x2}],
  {x2, 4, 0, -1}, {x1, 0, 5}]) // MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 4 & 4 & 5 & 5 \\ 0 & 0 & 4 & 4 & 4 & 5 \\ 0 & 0 & 3 & 3 & 3 & 3 \\ 0 & 0 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

```
(ubptabr =
  Table[SystemStateFromUBP[ubpsreal,fphi,{x1,x2}],
  {x2, 4, 0, -1}, {x1, 0, 5}]) // MatrixForm
```

$$\begin{pmatrix} 3 & 3 & 4 & 4 & 5 & 5 \\ 3 & 3 & 4 & 4 & 4 & 5 \\ 3 & 3 & 3 & 3 & 3 & 5 \\ 2 & 2 & 2 & 2 & 2 & 5 \\ 0 & 0 & 1 & 1 & 1 & 5 \end{pmatrix}$$

```
phitab==lbptabr
```

```
False
```

```
phitab==ubptabr
```

```
False
```

As we can see, incorrect results are produced when the "Virtual" boundary points are not included in the boundary point lists.

To further test this software package, we verify that we can obtain the upper boundary points based on the lower boundary points, and visa versa:

```
(Drop[#,-1]& /@ LBPToUBP[lbps,p,fphi]) ==
(Drop[#,-1]& /@ ubps)
```

```
True
```

```
(Drop[#,-1]& /@ UBPToLBP[ubps,p,fphi]) ==
(Drop[#,-1]& /@ lbps)
```

```
True
```

Now, we can test if all the components are relevent, based not on the structure function but instead on the boundary points.

```
BoedigheimerReleventComponentsQ[lbps,ubps,p]
```

```
True
```

Although this is not generally necessary to do for boundary points which are calculated from a coherent system instead of being provided by the customer, we can check for the internal and relative consistency of the boundary point sets:

```
LBPSelfConsistentQ[lbps]
```

```
True
```

```
UBPSelfConsistentQ[ubps]
```

```
True
```

```
BPConsistentToEachOtherQ[lbps,ubps]
```

```
True
```

Although we would not expect this to be the case due to the rather arbitrary nature of this structure function, we can check, using the Boedigheimer[1992] definitions for series and parallel based on the boundary points, whether this system fits either set of criteria.

```
BoedigheimerParallelQ[lbps,ubps,fphi]
```

```
False
```

```
BoedigheimerSeriesQ[lbps,ubps,fphi]
```

```
False
```

We can also find the structural importances of the components.

```
StructuralImportances[p,phi]
```

$$\left\{1, \frac{5}{6}\right\}$$

We can find the system extrema based on the boundary point sets (a trivial problem):

```
SystemLimitsFromBP[lbps,ubps]
```

```
{0, 5}
```

We can also find the full fphi set based on the boundary points (another trivial problem):

```
SystemSpaceFromBP[lbps,ubps]
```

```
{0, 1, 2, 3, 4, 5}
```

■ Stochastic Behavior of the System

To proceed, we must have a matrix which represents the probabilities that each component is in each of its possible states as a function of time. Since stochastic information was not given in the original formulation of this problem, we arbitrarily assign probabilities. For this section, we will assign static probabilities which are independent of time.

```
pprob = {{0.1,0.2,0.3,0.2,0.15,0.05},
         {0.2,0.2,0.3,0.1,0.2}}
```

```
{{0.1, 0.2, 0.3, 0.2, 0.15, 0.05}, {0.2, 0.2, 0.3, 0.1, 0.2}}
```

The first sublist gives the probabilities that x1 is in each one of its states, while the second sublist gives the probabilities that x2 is in each one of its states. To be sure that we have not entered data incorrectly, we can call a function which will check that our probability matrix is the same size as p, and that the sum of each sublist is one.

```
ConsistentProbabilitiesQ[p, pprob]
```

```
True
```

We can easily calculate the exact probabilities of the system being in any one of its states, assuming that the components are mutually indpendent, through direct enumeration.

```
ans=SystemFromDirectEnumeration[p, phi, fphi, pprob]
```

```
{0.18, 0.14, 0.14, 0.33, 0.165, 0.045}
```

We can also calculate the "reliability importances" (also called "performance importances") of any of the components. This is the stochastic analog of the structural importance that we calculated earlier. First, we calculate it for level 3 of component 1 (x1):

```
ReliabilityImportance[p, phi, fphi, pprob, 1, 3]
```

```
2.7
```

Now, we call another function which will return the complete table of reliability importances, for every level of every component. In general, we may wish to direct attention for improvements (all other things being equal) to components that exhibit a high reliability importance in the state range where they are expected to reside.

```
ReliabilityImportancesTable[p, phi, fphi, pprob]
```

```
{{0, 1.8, 2.7, 2.7, 2.9, 3.}, {0., 0.7, 2., 2.75, 2.9}}
```

■ Dynamic Behavior of the System

Since no stochastic or dynamic information was given in the original formulation of this problem, we will hypothesize a dynamic model and explore its characteristics before proceeding with the reliability measures analysis.

■ PDPErlangian[M,val,t]

Let us begin by assuming that each component i begins in its maximal state and spends a length of time in each state governed by an exponential distribution with a common parameter mui before progressing to the next lower state. We assume that the components no longer change state once they reach their minimal states. We wish to calculate for such a model the probability that the components are in any one of their given states as a function of time, as that is the information which is required by the reliability measures and system state calculation algorithms.

We can calculate these dynamic probabilities for each of the components as follows. The lists which follow the calculation for each component consist of the probabilities as a function of time that each component is in any one of its states. The second element in the first list, for example, is the probability that component x1 in its second-to-worst state.

```
x1prob = PDPErlangian[5,mu1,t] // Simplify
```

$$\left\{\frac{1}{24}\,E^{-mu1\,t}\,\left(24\,\left(-1+E^{mu1\,t}\right)-24\,mu1\,t-12\,mu1^2\,t^2-4\,mu1^3\,t^3-mu1^4\,t^4\right),\right.$$
$$\left.\frac{1}{24}\,E^{-mu1\,t}\,mu1^4\,t^4,\ \frac{1}{6}\,E^{-mu1\,t}\,mu1^3\,t^3,\ \frac{1}{2}\,E^{-mu1\,t}\,mu1^2\,t^2,\ E^{-mu1\,t}\,mu1\,t,\ E^{-mu1\,t}\right\}$$

```
x2prob = PDPErlangian[4,mu2,t] // Simplify
```

$$\left\{\frac{1}{6} E^{-\text{mu2}\, t} \left(-6+6 E^{\text{mu2}\, t}-6\, \text{mu2}\, t-3\, \text{mu2}^2\, t^2-\text{mu2}^3\, t^3\right),\ \frac{1}{6} E^{-\text{mu2}\, t}\, \text{mu2}^3\, t^3,\ \frac{1}{2} E^{-\text{mu2}\, t}\, \text{mu2}^2\, t^2,\ E^{-\text{mu2}\, t}\, \text{mu2}\, t,\ E^{-\text{mu2}\, t}\right\}$$

Now, using direct enumeration, we can calculate the probability that the SYSTEM is in any one of its possible states, based on the structure function and the probability of each of the components being in any one of its possible states.

First, we form a matrix in the form needed:

```
pprob1={x1prob, x2prob};
```

And then calculate the result:

```
ans1=SystemFromDirectEnumeration[p, phi, fphi, pprob1] //
  Simplify
```

$$\left\{\frac{1}{48} E^{-(\text{mu1}+\text{mu2})\, t} \left(48\, E^{\text{mu2}\, t} \left(-1+E^{\text{mu1}\, t}\right)-48\, E^{\text{mu2}\, t}\, \text{mu1}\, t-24\, E^{\text{mu2}\, t}\, \text{mu1}^2\, t^2-8\, E^{\text{mu2}\, t}\, \text{mu1}^3\, t^3-\text{mu1}^4\, t^4 \left(2+2\, \text{mu2}\, t+\text{mu2}^2\, t^2\right)\right),\right.$$
$$-\frac{1}{36} E^{-(\text{mu1}+\text{mu2})\, t} \left(6+6\, \text{mu1}\, t+3\, \text{mu1}^2\, t^2+\text{mu1}^3\, t^3\right) \left(6-6\, E^{\text{mu2}\, t}+6\, \text{mu2}\, t+3\, \text{mu2}^2\, t^2+\text{mu2}^3\, t^3\right),$$
$$\frac{1}{36} E^{-(\text{mu1}+\text{mu2})\, t}\, \text{mu2}^3\, t^3 \left(6+6\, \text{mu1}\, t+3\, \text{mu1}^2\, t^2+\text{mu1}^3\, t^3\right),$$
$$\frac{1}{48} E^{-(\text{mu1}+\text{mu2})\, t}\, t^2 \left(2\, \text{mu1}^4\, t^2+2\, \text{mu1}^4\, \text{mu2}\, t^3+\text{mu2}^2 \left(24+24\, \text{mu1}\, t+12\, \text{mu1}^2\, t^2+4\, \text{mu1}^3\, t^3+\text{mu1}^4\, t^4\right)\right),$$
$$\frac{1}{6} E^{-(\text{mu1}+\text{mu2})\, t}\, \text{mu1}\, t^2 \left(6\, \text{mu2}+3\, \text{mu1} \left(1+\text{mu2}\, t\right)+\text{mu1}^2\, t \left(1+\text{mu2}\, t\right)\right),$$
$$\left.E^{-(\text{mu1}+\text{mu2})\, t} \left(1+\text{mu1}\, t+\text{mu2}\, t\right)\right\}$$

In real life, we would perform a series of tests (see Kapur and Lamberson [1977]) to determine the values of mu1 and mu2. Here we will arbitrarily assume that mu1 is 1.2 and mu2 is 1.9. This causes the above expression to read:

```
ans1 = ans1 /. {mu1->1.2, mu2->1.9} // Simplify
```

$$\{1.\, E^{-3.1\, t} \left(-1.\, E^{1.9\, t}+1.\, E^{3.1\, t}-1.2\, E^{1.9\, t}\, t-0.72\, E^{1.9\, t}\, t^2-0.288\, E^{1.9\, t}\, t^3-0.0864\, t^4-0.16416\, t^5-0.155952\, t^6\right),$$
$$-0.048\, E^{-3.1\, t} \left(1.33006+t\right) \left(2.61058+1.16994\, t+t^2\right) \left(6-6\, E^{1.9\, t}+11.4\, t+10.83\, t^2+6.859\, t^3\right),$$
$$0.329232\, E^{-3.1\, t}\, t^3 \left(1.33006+t\right) \left(2.61058+1.16994\, t+t^2\right),$$
$$0.155952\, E^{-3.1\, t}\, t^2 \left(3.08605+0.532163\, t+t^2\right) \left(3.75044+3.8538\, t+t^2\right),$$
$$0.5472\, E^{-3.1\, t}\, t^2 \left(5.48246+3.02632\, t+1.\, t^2\right),\ E^{-3.1\, t} \left(1+3.1\, t\right)\}$$

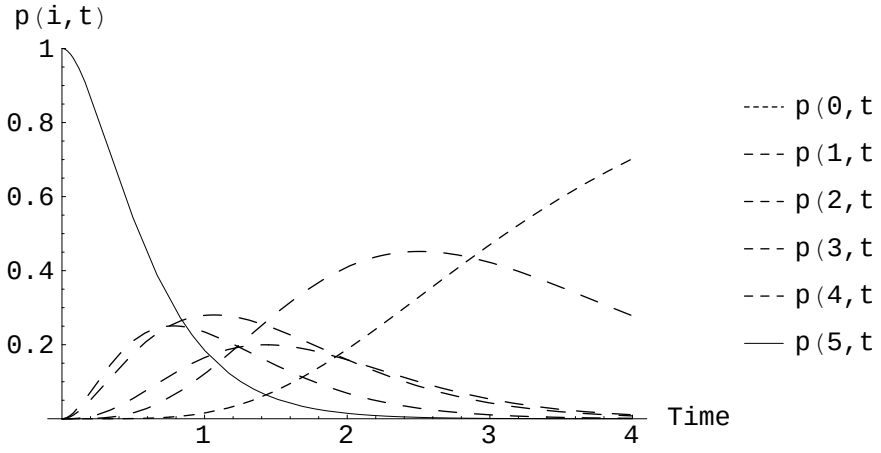
We can verify that the probability of the system being in SOME state is one at least at one point in time (we could draw a graph or solve analytically to convince ourselves of this relation for all non-negative t):

```
Plus @@ ans1 /. t-> 3
```

```
1.
```

Now, let's graph the probabilities of the system being in any one of its possible states.

```
Plot[Evaluate[ans1], {t,0,4},
  PlotRange->{0,1},
  AxesLabel->{"Time","p(i,t)"},
  PlotStyle->{Dashing[{0.017}],
              Dashing[{0.03}],
              Dashing[{0.03}],
              Dashing[{0.03}],
              Dashing[{0.03}],
              Thickness[0.001]},
  PlotLegend->{"p(0,t)", "p(1,t)", "p(2,t)",
               "p(3,t)", "p(4,t)", "p(5,t)"},
  LegendPosition->{1,-0.4}, LegendShadow->None];
```



■ Measures

```
ans1sys = Transpose[{fphi, ans1}];
```

```
ans1sys /. t -> 2 // MatrixForm
```

$$\begin{pmatrix} 0 & 0.187555 \\ 1 & 0.410009 \\ 2 & 0.159317 \\ 3 & 0.159499 \\ 4 & 0.0690071 \\ 5 & 0.0146119 \end{pmatrix}$$

```
ExpectedState[ans1sys /. t -> 2]
```

```
1.55623
```

```
ExpectedState[ans1sys /. t -> 2] / Max[fphi]
```

```
0.311246
```

```
StateVariance[ans1sys /. t -> 2]
```

```
1.53033
```

```
ExpectedTotalOutput[ans1sys, t, True]
```

```
8.45831
```

```
ExpectedOutput[ans1sys, t, 2, True]
```

```
6.46482
```

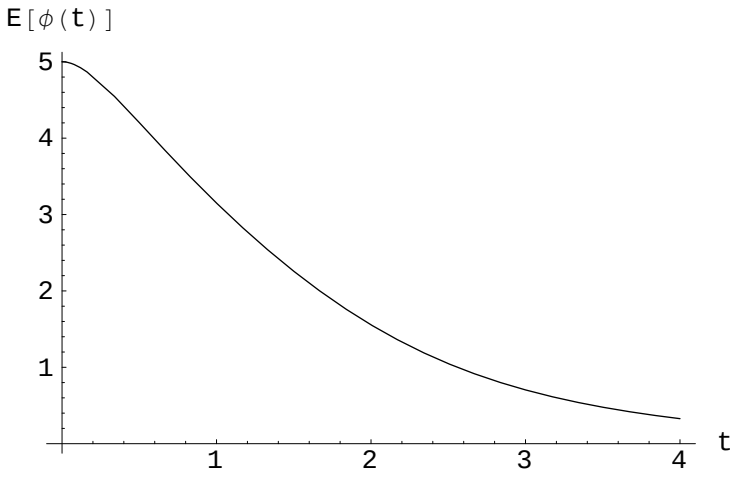
```
VarianceOfOutputUB[ans1sys, t, 2, True]
```

```
6.55507
```

```
ExpectedOutput[ans1sys, t, 2, True] /
 ExpectedTotalOutput[ans1sys, t, True]
```

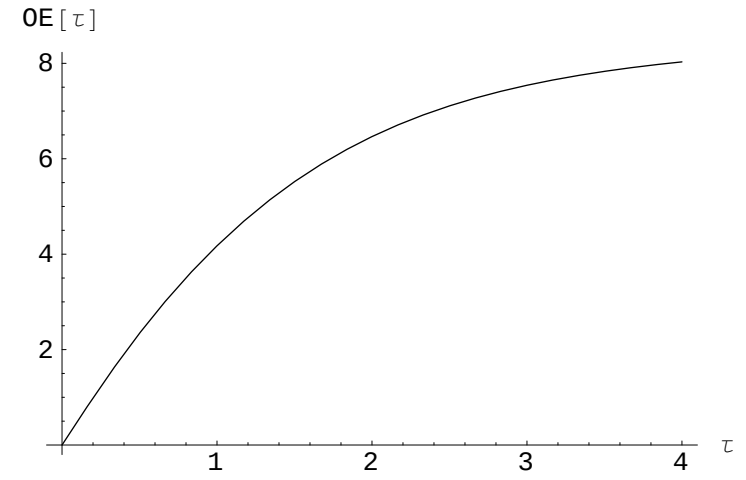
```
0.764316
```

```
Plot[ExpectedState[ans1sys], {t, 0, 4},
 AxesLabel -> {"t", "E[ϕ(t)]"}]
```



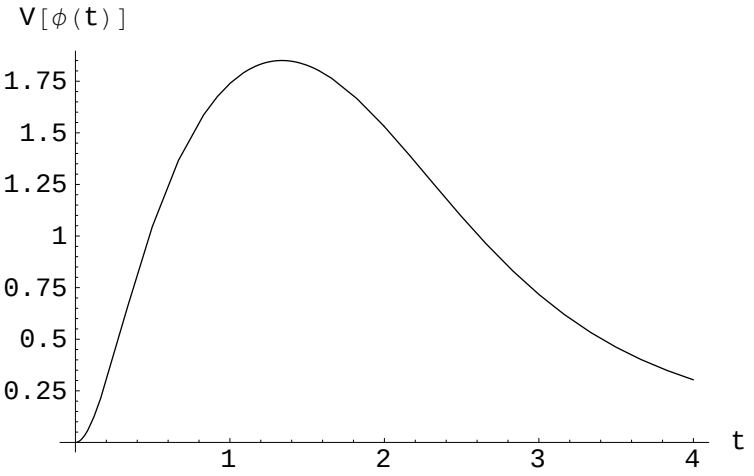

```
- Graphics -
```

```
Plot[ExpectedOutput[ans1sys, t, τ, True], {τ, 0, 4},
 AxesLabel -> {"τ", "OE[τ]"}]
```




```
- Graphics -
```

```
Plot[StateVariance[ans1sys], {t, 0, 4},
 AxesLabel -> {"t", "V[ϕ(t)]"}]
```




```
- Graphics -
```

```
Table[UpperStatesDwellTime[ans1sys, t, j, True], {j, 0, 5}]
```

```
{3.40358, 1.89813, 1.51141, 1.00002, 0.645161, 0.}
```

```
LifetimeWeighted[ans1sys, t, 2 E^(-2 t)]
```

```
4.19442
```

```
LifetimeWeighted[ans1sys, t, 9 3^(1/2) (3 + 2 t)^(-5/2)]
```

```
2.92282
```

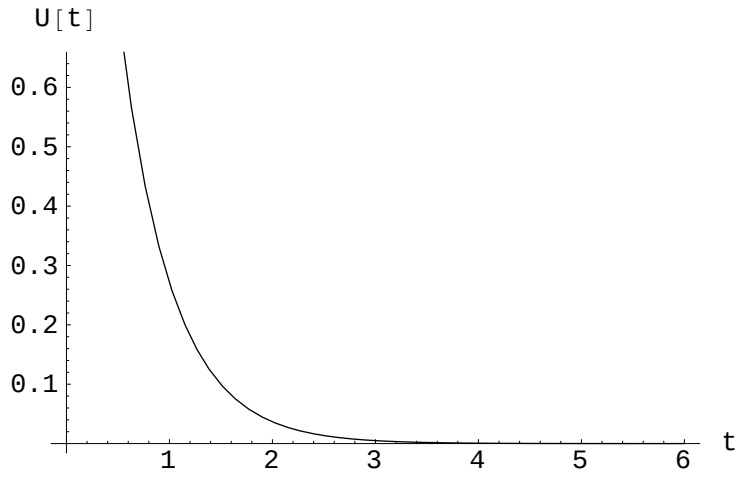
```
LifetimeWeighted[ans1sys, t, (t/9) E^(-t/3)]
```

```
0.627763
```

```
LifetimeWeighted[ans1sys, t, 2 t E^(-t^2)]
```

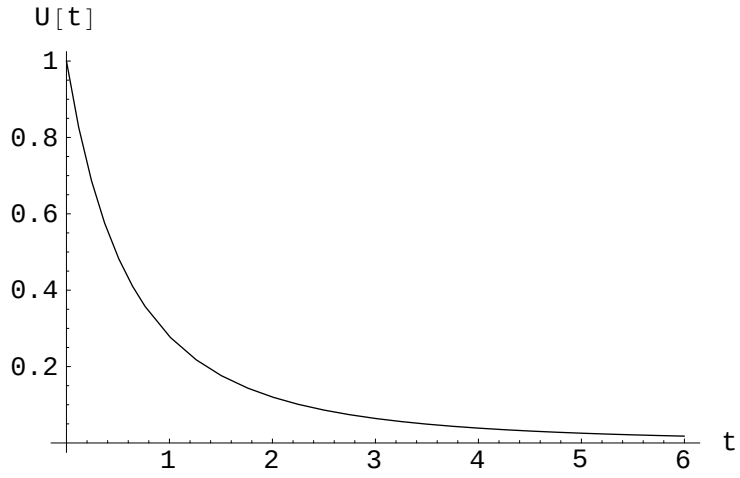
```
3.43514
```

```
Plot[2 E^(-2 t), {t, 0, 6}, AxesLabel -> {"t", "U[t]"}]
```



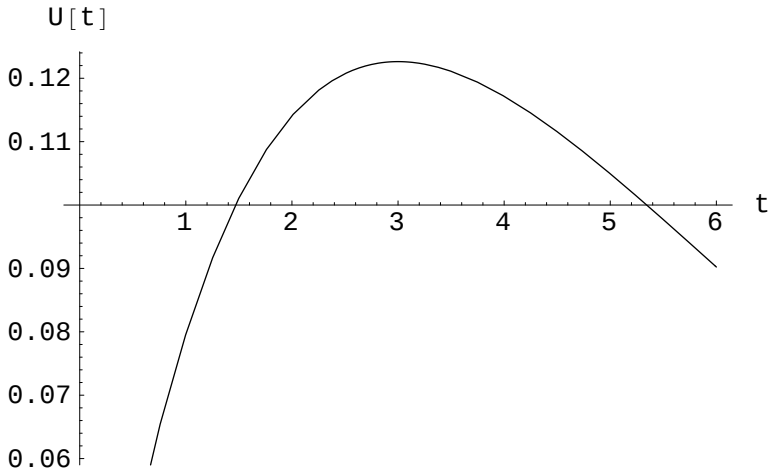

```
- Graphics -
```

```
Plot[9 3^(1/2) (3 + 2 t)^(-5/2), {t, 0, 6},
 AxesLabel -> {"t", "U[t]"}]
```



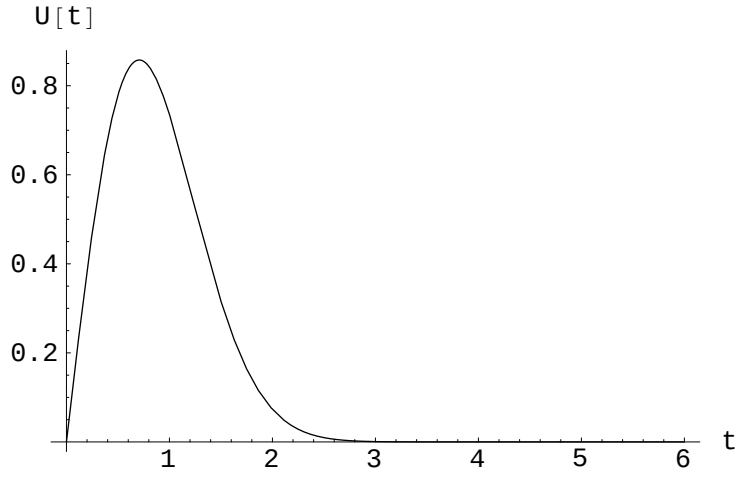

```
- Graphics -
```

```
Plot[(t / 9) E^(-t / 3), {t, 0, 6}, AxesLabel -> {"t", "U[t]"}]
```



- Graphics -

```
Plot[2 t E^(-t^2), {t, 0, 6}, AxesLabel -> {"t", "U[t]"}]
```



- Graphics -

Mixed Model Tutorial

■ Introduction

This document considers the general analysis of mixed systems. These may have non-zero probabilities of being in any given state, superimposed over a continuous distribution. To illustrate the basic techniques involved, this document sometimes uses full *Mathematica* code to solve problems rather than calling a RelPack function to do that particular task. The idea is to make certain simple functions more understandable to the user by making their functionality transparent.

■ Sample Problem 1

First, we define a list containing the CDF's for the state of each of the components:

```
distlist={BinaryCDF[x,0.5],UniformMixedCDF[x,0.2,0.3],
          TruncatedCDF[x,ChiDistribution[1]]};
```

Here are the values of each of these CDF's at x=0.1. Essentially, this is a list of the probabilities that random variable drawn from each of these distributions would be 0.1 or lower.

```
distlist /. x-> 0.1
```

```
{0.5, 0.25, 0.0796557}
```

Supposing that these distributions modeled the random variable representing the state of a component, and that a system was composed of a "parallel" (maximum) arrangements of these components, the following function returns the probability that the system is in or below state 0.1

```
P[distlist,x->.1]
```

```
0.00995696
```

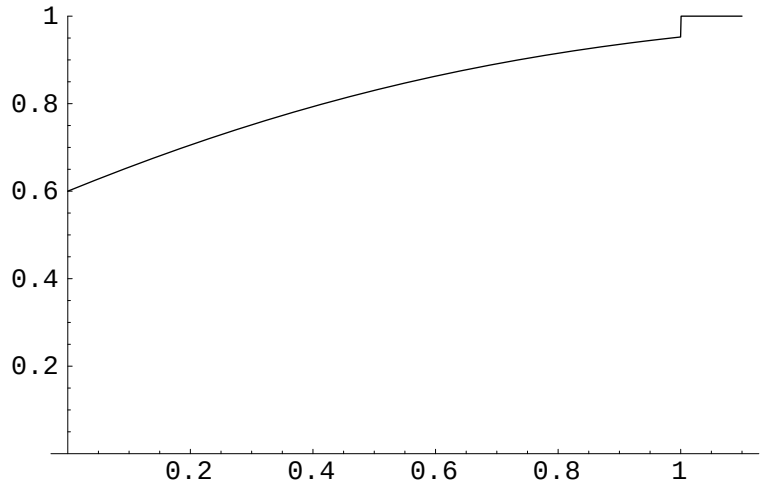
Assuming a "series" (minimum) arrangement of these components in the system, we compute the probability of being in or below state 0.1 as follows. As we would expect, the probability of being in lower states is higher for a series system than for a parallel system.

```
S[distlist,x->.1]
```

```
0.654871
```

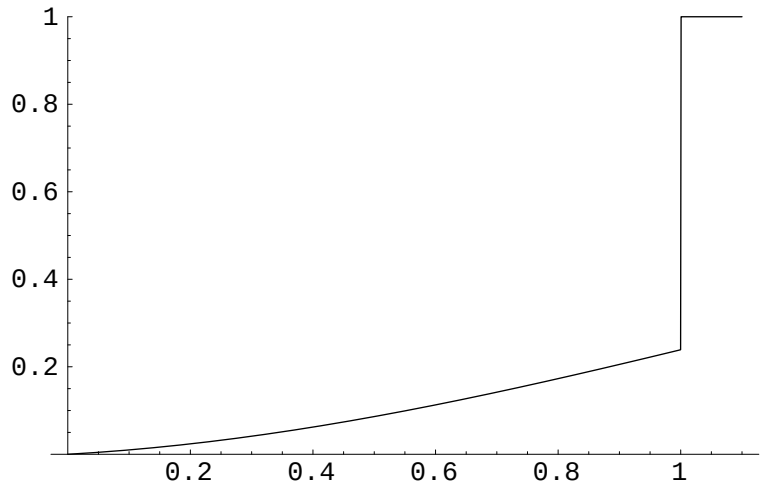
Here is a graph of the CDF of the system in the series case:

```
Plot[S[distlist],{x,0,1.1},PlotRange->{0,1}];
```



Here is a graph of the CDF of the system in the parallel case:

```
Plot[P[distlist],{x,0,1.1},PlotRange->{0,1}];
```



Here are the values of all the component CDF's at x=0 and x=1, respectively:

```
{distlist /. x-> 0, distlist /. x-> 1}
```

```
{{0.5, 0.2, 0}, {1, 1, 1}}
```

Here is the value of the parallel system CDF at x=0 and x=1, respectively:

```
{P[distlist,x->0], P[distlist,x->1]}
```

```
{0, 1}
```

Here is the value of the series system CDF at x=0 and x=1, respectively:

```
{S[distlist,x->0], S[distlist,x->1]}
```

```
{0.6, 1}
```

We can take advantage of our knowledge of the particular distributions that we have selected. Since we know that the probability of the system being in any particular state is zero except for states in the set {0,1}, we can proceed for the series case and find the probabilities of the system being in either its maximum or minimum state:

```
ans=S[distlist];
```

```
{ans /. x-> 0, 1-(ans /. x-> (1-$MachineEpsilon))}
```

```
{0.6, 0.0475966 }
```

■ Reduction to Binary Results

As we can see, the methodology we have used above reduces to the standard binary result for parallel and series systems:

```
distlist2 = {BinaryCDF[x,0.4], BinaryCDF[x,0.8],
             BinaryCDF[x,0.9]};
```

```
1-P[distlist2,x->0]
```

```
0.988
```

```
1-(1-0.4)*(1-0.8)*(1-0.9)
```

```
0.988
```

The first expression gave us the probability of being in state 0, which is why we had to subtract one from it to get the probability of being in state one. Of course, 1-(1-p1)(1-p2)(1-p3) is the well known result for the probability of a binary parallel system being in state 1 when the probability of each of its components being in state 1 is pi.

Here is the analogous demonstration for series systems.

```
1-S[distlist2,x->0]
```

```
0.288
```

```
0.4*0.8*0.9
```

```
0.288
```

■ Sample Problem 2

This problem will be considerably more complex. We will consider a system with 20 components, each with a different distribution. Some of the distributions will be continuous, some will be discrete, some will be binary, and some will be mixed. Please see attached sheet for a diagram of this system. Note that the reliability block diagram form for "parallel" is used when the structure function module for those components is Maximum, and the reliability block diagram form for "series" is used when the structure function module for those components is Minimum.

Here is the list containing the distributions of the components:

```
d={BinaryCDF[x,0.5],
   UniformMixedCDF[x,0.2,0.3],
   TruncatedCDF[x,ChiDistribution[1]],
   BinaryCDF[x,0.432],
   BinaryCDF[x,0.643],
   UniformMixedCDF[x,0.05,0.48],
   TruncatedCDF[x,NormalDistribution[0.52,0.2]],
   TruncatedCDF[x,BetaDistribution[0.7,0.31]],
   TruncatedCDF[x,CauchyDistribution[0.3,2]],
   TruncatedCDF[x,ChiSquareDistribution[1]],
   TruncatedCDF[x,ExponentialDistribution[2]],
   TruncatedCDF[x,ExtremeValueDistribution[0.6,3]],
   TruncatedCDF[x,FRatioDistribution[5,7]],
   TruncatedCDF[x,GammaDistribution[0.8,0.3]],
   TruncatedCDF[x,LaplaceDistribution[0.2,0.238]],
   TruncatedCDF[x,LogNormalDistribution[0.2,1]],
   TruncatedCDF[x,LogisticDistribution[0.222,0.6]],
   TruncatedCDF[x,RayleighDistribution[1]],
   TruncatedCDF[x,StudentTDistribution[4]],
   TruncatedCDF[x,WeibullDistribution[.4,.9]]};
```

Here we define the structure function for the system, breaking it up into three modules for clarity:

```
moda=S[{P[{P[{S[{d[[1]],d[[2]]}],d[[3]]}],
         S[{P[{d[[6]],d[[7]]}],d[[4]],d[[5]]}]}],
        d[[8]]}];

modb=S[{d[[18]],P[{d[[19]],d[[20]]}]}];

modc=S[{P[{d[[16]],d[[17]]}],d[[15]]}];

modd=S[{P[{S[{d[[12]],d[[13]]}],d[[11]],d[[14]]}],
        d[[9]],d[[10]]}];

sys=P[{modb,S[{modc,P[{modd,moda}]}]}];
```

Here is the computed CDF for the system, based on the above distributions and structure function:

```
sys=(1-(1-TruncatedCDF[x,RayleighDistribution[1]])*
     (1-TruncatedCDF[x,StudentTDistribution[4]]*
        TruncatedCDF[x,WeibullDistribution[0.4,0.9]]))*
  (1-(1-TruncatedCDF[x,
        LaplaceDistribution[0.2,0.238]])*
     (1-TruncatedCDF[x,LogisticDistribution[0.222,0.6]]*
        TruncatedCDF[x,LogNormalDistribution[0.2,1]])*
     (1-(1-(1-TruncatedCDF[x,
              CauchyDistribution[0.3,2]])*
           (1-TruncatedCDF[x,ChiSquareDistribution[1]])*
           (1-TruncatedCDF[x,ExponentialDistribution[2]]*
              (1-(1-TruncatedCDF[x,
                    ExtremeValueDistribution[0.6,3]])*
                 (1-TruncatedCDF[x,
                    FRatioDistribution[5,7]]))*
              TruncatedCDF[x,GammaDistribution[0.8,0.3]]))*
        (1-(1-TruncatedCDF[x,
              BetaDistribution[0.7,0.31]])*
           (1-TruncatedCDF[x,ChiDistribution[1]]*
              (1-(1-BinaryCDF[x,0.4320000000000001])*
                 (1-BinaryCDF[x,0.643])*
                 (1-TruncatedCDF[x,
                     NormalDistribution[0.5200000000000001,
                     0.2]]*UniformMixedCDF[x,0.05,0.48]))*
              (1-(1-BinaryCDF[x,0.5])*
                 (1-UniformMixedCDF[x,0.2,0.3])))))));
```

The Probability the System is in State 0:

```
sys /. x->0
```

```
0
```

The Probability the System is in State 1

```
1-(sys /. x->(1-$MachineEpsilon))
```

```
0.287569
```

The Probability that the Components are in State 0

```
N[d /. x->0,5]
```

```
{0.5, 0.2, 0, 0.568, 0.357, 0.05, 0.0046612, 0, 0.45261, 0,
 0, 0.29482, 0, 0, 0.21578, 0, 0.40854, 0, 0.5, 0}
```

The Probabibility that the Components are in State 1

```
N[1-(d /. x->(1-$MachineEpsilon)),5]
```

```
{0.5, 0.3, 0.31731 , 0.432 , 0.643 , 0.48 , 0.0081975 ,
 0.000012004 , 0.39283 , 0.31731 , 0.13534 , 0.58321 , 0.48129 ,
 0.022972 , 0.017344 , 0.57926 , 0.21473 , 0.60653 , 0.18695 , 0.35238 }
```

Here is a list of the components which have a zero probability of being in state 0. It should be noted that if the system diagram is thought of as being one for a binary system, and if all the components in any minimal cut are such that the probability of those components being in state 0 is 0, then the probability that the system is in state 0 is 0.

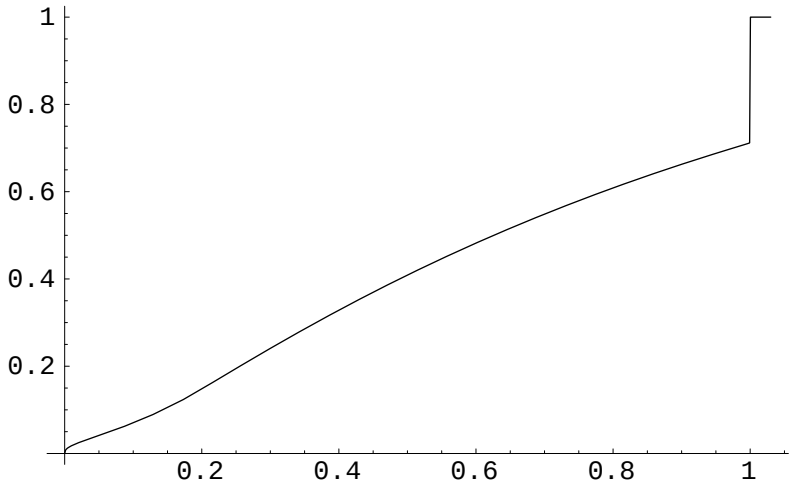
```
Flatten[Position[d /. x->0, 0]]
```

```
{3, 8, 10, 11, 13, 14, 16, 18, 20}
```

Note from the attached diagram that one minimal cut set to this system, if we are thinking of it in a binary sense, is the set {8, 10, 18}. The probabilities that each of these components is 0 is 0, so the probability that the system is in state 0 is 0.

Here is a plot of the CDF for this system:

```
Plot[sys,{x,0,1.03}];
```

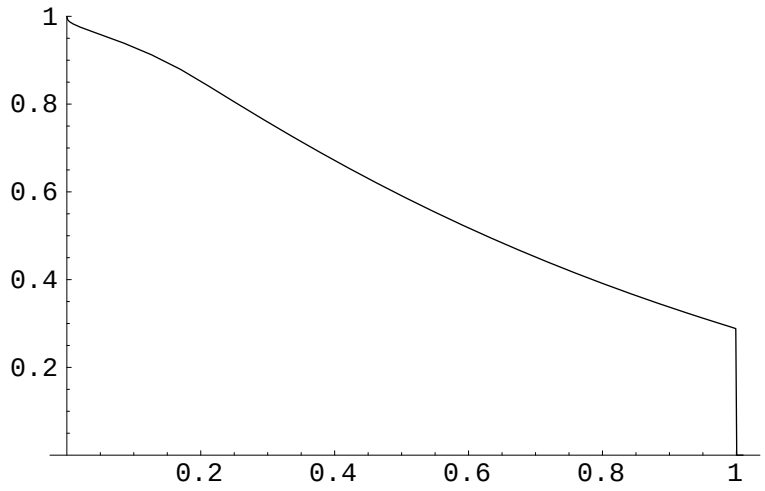


Here is a plot of 1-CDF, the "Reliability" for this system at this particular moment in time. Note the non-zero probability of being in state 1, as evidenced by the discontinuity in the graph at x=1.

■ Reliability Measures for Sample Problem 2

Now that we have the CDF for the system, any number of reliability measures may be calculated for the system. We start with the probability of the system being in or above any particular state:

```
Plot[1-sys,{x,0,1.01},PlotRange->{0,1}];
```



It would also be helpful to define a function which returns R(x) for any value of x:

```
RSystem[cdf_, rule_] := (1-cdf) /. rule
```

```
RSystem[sys, x->0.431]
```

```
0.645831
```

```
CDFMean[cdf_, x_] := NIntegrate[1-cdf,{x,0,Infinity}]
```

And compute the result.

```
CDFMean[sys,x]
```

```
0.613852
```

Thus, we find that the expected state of the system at this moment in time is 61% of its maximal state.

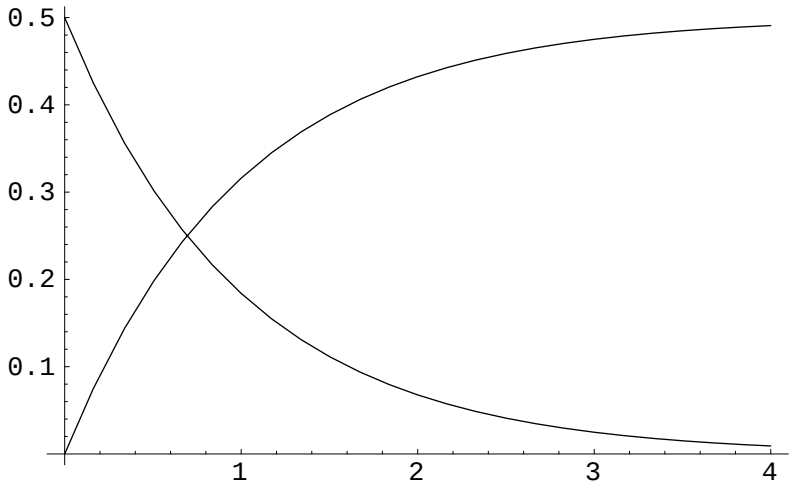
■ Dynamic Systems

Of course, any of these techniques may be applied to dynamic systems, for which the CDF's for each of the components is some function of time. One need only use the desired time based model. Let us assume a model for which the probability of being in the maximal state is some non-zero value (for each component), the probabibility of being in the minimal state is some non-zero value, and the remaining probability is evenly distributed between 0 and 1. We will assume that the probability of the system being in its minimal state exponentially increases, and the probability of the system being in its maximal state gradually increases. We will assume a decay rate of 1 for the component 1, 2 for component 2, and 3 for component 3. The probability of being in either of the two discrete states is 0.5.

```
d2={UniformMixedCDF[x,0.5(1-E^(-1t)),0.5E^(-1t)],
    UniformMixedCDF[x,0.5(1-E^(-2t)),0.5E^(-2t)],
    UniformMixedCDF[x,0.5(1-E^(-3t)),0.5E^(-3t)]};
```

Let's graph the probabilities of being in either of the two discrete states for component one as a function of time:

```
Plot[Evaluate[{0.5 (1-E^(-1t)),0.5 E^(-1t)}],{t,0,4}];
```

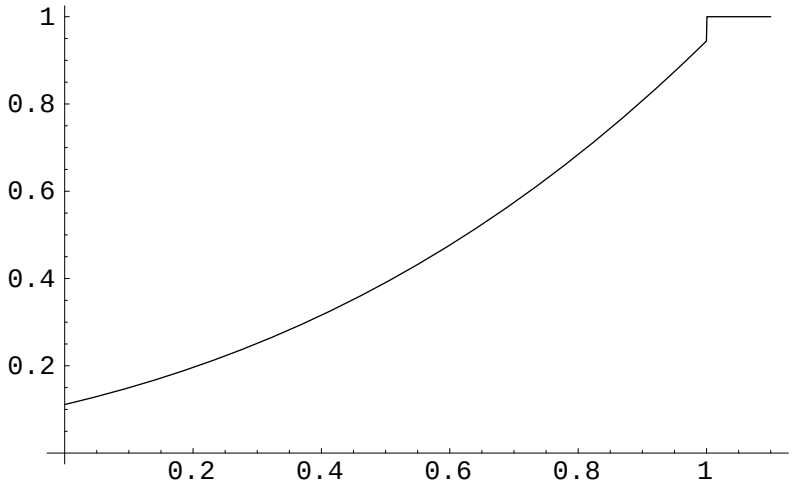


Let's consider a parallel system of these components:

```
sys2=P[d2];
```

Let's examine the system CDF at a particular moment in time, let's say at t=2.3. Note the non-zero probabilities of the system being in either state 1 or state 2.

```
Plot[sys2 /. t->2.3, {x, 0, 1.1}];
```



Let'sask ourselveswhat the probability is of each of the componentsbeing in either of the discrete states at t=2.3:

The Probabibilitythat the Componentsare in State 0

```
N[d2 /. {x->0,t->2.3},5]
```

```
{0.44987 , 0.49497 , 0.4995 }
```

The Probabibilitythat the Componentsare in State 1

```
N[1-(d2 /. {x->(1-$MachineEpsilon),t->2.3}),5]
```

```
{0.050129 , 0.0050259 , 0.00050389 }
```

Since the system is in state 0 if all of the componentsis in state 0, we would predict the following as the probability of the system being in state 0 at this time:

```
Times @@ N[d2 /. {x->0,t->2.3},5]
```

```
0.111225
```

And since the system is in state 1 if any of the componetsare in state 1, we would predict the followingas the probability of the state of the system being 1 at this time:

```
1-(Times @@ (1-N[1-(d2 /. {x->(1-$MachineEpsilon),
   t->2.3}),5]))
```

```
0.0553796
```

Calculated from the computed CDF (note that the values are identical):

The Probability the System is in State 0:

```
sys2 /. {x->0,t->2.3}
```

```
0.111225
```

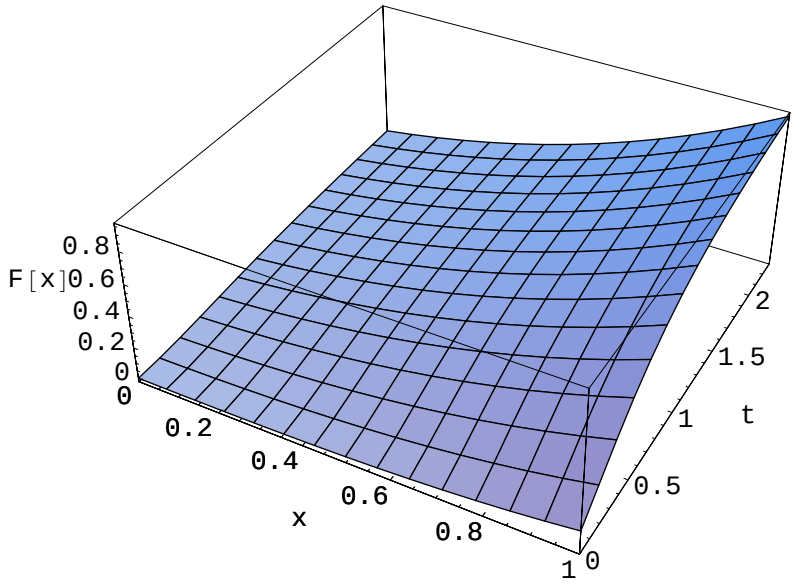
The Probability the System is in State 1

```
1-(sys2 /. {x->(1-$MachineEpsilon),t->2.3})
```

```
0.0553796
```

Let's examine a three dimensional plot of the CDF of this system as a function of time. Note that the final "jump" of this curve at x=1 has been omitted for clarity.

```
Plot3D[sys2, {x,0,1},{t,0,2.3},
       AxesLabel->{"x","t","F[x]"}];
```



Here is the Reliability of the system at level x=0.431 and t=2.3

```
RSystem[sys2, {x->0.431,t->2.3}]
```

```
0.662001
```

And here is the expected state of the system at that time:

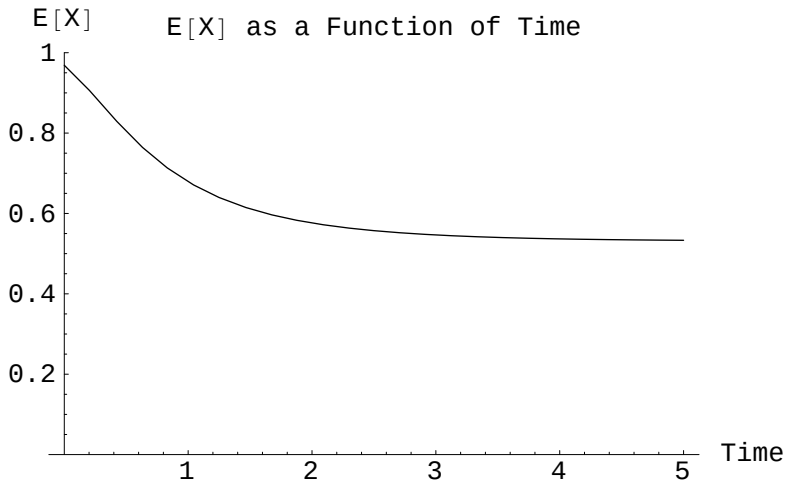
```
CDFMean[sys2 /. t->2.3,x]
```

```
0.563508
```

Thus, we find that the expected state of the system at this moment in time is 56% of its maximal state at t=2.3.

Now, let's graph the expected state of the system as a function of time:

```
Plot[CDFMean[sys2 /. t->t0,x], {t0,0,5},
     PlotLabel->"E[X] as a Function of Time",
     PlotRange->{0,1},
     AxesLabel->{"Time","E[X]"}]
```




```
- Graphics -
```

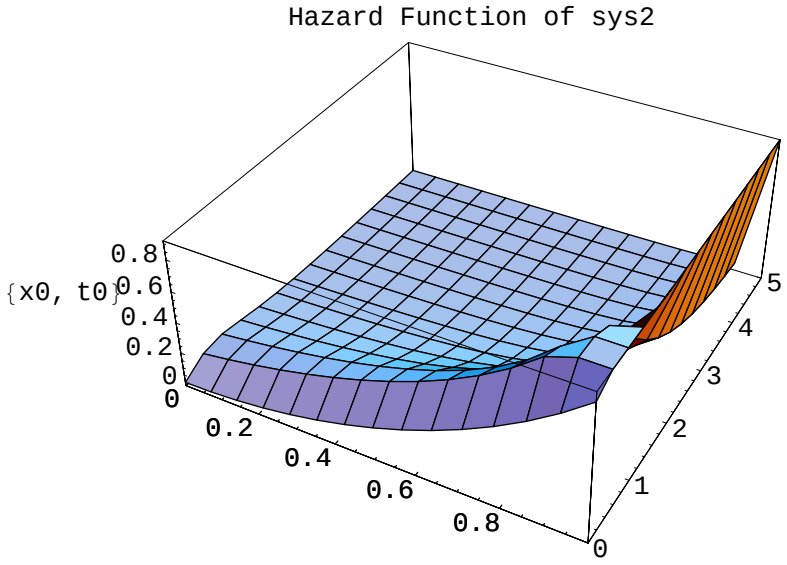
As this graph leads one to believe, the mean state of this system approaches some value asymptotically as t->Infinity. This limiting value may be computed directly as follows:

```
CDFMean[P[Table[UniformMixedCDF[x,0.5,0],{3}]],x]
```

```
0.53125
```

Here is a plot of the hazard function for this dynamic system, as a function of time:

```
Plot3D[CDFHazard[sys2, x, x0, t, t0], {x0,0,0.99999},
   {t0, 0, 5}, PlotLabel->"Hazard Function of sys2",
               AxesLabel->{"x0","t0"}]
```




```
- SurfaceGraphics -
```

Automobile Example

■ New Function Definition

```
discrete[sys_,fphi_,phi_] :=
Module[{ans,gfphi,gpprob,finans},
ans={GenAns[(sys[0][[1]]),x1,fphi],
     GenAns[(sys[0][[2]]),x2,fphi],
     GenAns[(sys[0][[3]]),x3,fphi],
     GenAns[(sys[0][[4]]),x4,fphi]};
gfphi=Transpose[First[ans]][[1]];
gpprob=Transpose[#][[2]]& /@ ans;
finans={gfphi,SystemFromDirectEnumeration[
   {gfphi,gfphi,gfphi,gfphi},phi,gfphi,gpprob]};
```

■ System C

```
(* Definition *)

phiraw[x_] := Times @@ x

sys[t_] := {PDF[BetaDistribution[25/16,1],x1],
            PDF[BetaDistribution[25/16,1],x2],
            PDF[BetaDistribution[25/16,1],x3],
            PDF[BetaDistribution[25/16,1],x4]}

(* Binary {0,1} *)

phib[x_] := phiround[x,phiraw,{0,1}]

discrete[sys,{0,1/2,1},phib]

0.191404

(* Discrete {0,1/4,1/2,3/4,1} *)

phi[x_] := phiround[x,phiraw,{0,1/4,1/2,3/4,1}]

discrete[sys,{0,1/8,3/8,5/8,7/8,1},phi]

0.12204

(* Continuous *)
```

```
NIntegrate[x1 sys[0][[1]],{x1,0,1}]*
NIntegrate[x2 sys[0][[2]],{x2,0,1}]*
NIntegrate[x3 sys[0][[3]],{x3,0,1}]*
NIntegrate[x4 sys[0][[4]],{x4,0,1}]
```

```
0.138237
```

■ System D

```
(* Definition *)

phiraw[x_] := (Log[x[[1]]+1]/Log[2])*
                    ((E^x[[2]]-1)/(E-1))*
                 (x[[3]]^4)*(Sqrt[x[[4]]])// N

sys[t_] := {TruncatedPDF[x1,
               NormalDistribution[0.75,0.2]],
            TruncatedPDF[x2,
               NormalDistribution[0.75,0.2]],
            TruncatedPDF[x3,
               NormalDistribution[0.75,0.2]],
            TruncatedPDF[x4,
               NormalDistribution[0.75,0.2]]}

(* Binary {0,1} *)

phib[x_] := phiround[x,phiraw,{0,1}]

discrete[sys,{0,1/2,1},phib]
```

```
0.605048
```

```
(* Discrete {0,1/4,1/2,3/4,1} *)

phi[x_] := phiround[x,phiraw,{0,1/4,1/2,3/4,1}]

discrete[sys,{0,1/8,3/8,5/8,7/8,1},phi]
```

```
0.129745
```

```
(* Continuous *)

NIntegrate[(Log[x1+1]/Log[2]) sys[0][[1]],{x1,0,1}]*
NIntegrate[((E^x2-1)/(E-1)) sys[0][[2]],{x2,0,1}]*
NIntegrate[(x3^4) sys[0][[3]],{x3,0,1}]*
NIntegrate[(Sqrt[x4]) sys[0][[4]],{x4,0,1}]
```

```
0.131736
```

■ System E

```
(* Definition *)

phiraw[x_] :=  (Log[x[[1]]+1]/Log[2]/4)+
                   ((E^x[[2]]-1)/(E-1)/4)+
                   ((x[[3]]^10)/4)+
                   (Sqrt[x[[4]]]/4) // N

sys[t_] := {TruncatedPDF[x1,
              NormalDistribution[0.1,0.1]],
            TruncatedPDF[x2,
              NormalDistribution[0.1,0.1]],
            2 (1-x3),
            TruncatedPDF[x4,
              NormalDistribution[0.1,0.1]]}

(* Binary {0,1} *)

phib[x_] := phiround[x,phiraw,{0,1}]

discrete[sys,{0,1/2,1},phib]
```

1.06279×10^{-9}

```
(* Discrete {0,1/4,1/2,3/4,1} *)

phi[x_] := phiround[x,phiraw,{0,1/4,1/2,3/4,1}]

discrete[sys,{0,1/8,3/8,5/8,7/8,1},phi]
```

0.1194

```
(* Continuous *)

NIntegrate[(Log[x1+1]/Log[2]/4) sys[0][[1]],{x1,0,1}]+
NIntegrate[((E^x2-1)/(E-1)/4) sys[0][[2]],{x2,0,1}]+
NIntegrate[((x3^10)/4) sys[0][[3]],{x3,0,1}]+
NIntegrate[(Sqrt[x4]/4) sys[0][[4]],{x4,0,1}]
```

0.151749

■ System F

```
(* Definition *)

phiraw[x_] :=
   x[[1]]^2 x[[2]]^4 x[[3]]^(1/2) x[[4]]^(1/8)

sys[t_] := {TruncatedPDF[x1,
              WeibullDistribution[3/2,2]],
            TruncatedPDF[x2,
              WeibullDistribution[3,4]],
            TruncatedPDF[x3,
              WeibullDistribution[2,5]],
            TruncatedPDF[x4,
              WeibullDistribution[6,3]]}

(* Binary {0,1} *)

phib[x_] := phiround[x,phiraw,{0,1}]

discrete[sys,{0,1/2,1},phib]
```

```
0.388768
```

```
(* Discrete {0,1/4,1/2,3/4,1} *)

phi[x_] := phiround[x,phiraw,{0,1/4,1/2,3/4,1}]

discrete[sys,{0,1/8,3/8,5/8,7/8,1},phi]
```

```
0.144999
```

```
(* Continuous *)

NIntegrate[x1^2 sys[0][[1]],{x1,0,1}]*
NIntegrate[x2^4 sys[0][[2]],{x2,0,1}]*
NIntegrate[x3^(1/2) sys[0][[3]],{x3,0,1}]*
NIntegrate[x4^(1/8) sys[0][[4]],{x4,0,1}]
```

```
0.133113
```

Bicycle Example

■ Function Definitions

```
discrete[sys_,x1_,x2_,fphi_,time_,phi_] :=
Module[{ans,gfphi,gpprob,finans},
ans={GenAns[(sys[time][[1]]),x1,fphi],
     GenAns[(sys[time][[2]]),x2,fphi]};
gfphi=Transpose[First[ans]][[1]];
gpprob=Transpose[#][[2]]& /@ ans;
finans=Transpose[{gfphi,SystemFromDirectEnumeration[
   {gfphi,gfphi},phi3,gfphi,gpprob]}];

contavg[time_,sys_,phi_] := NIntegrate[phi[{x1,x2}]*
   (Times @@ sys[time]),{x1,0,1},{x2,0,1}]

sdcontavg[time_,sys_,data_] := Module[{datac},
   datac=MultiQuadricC[data];
NIntegrate[MultiQuadric[{x1,x2},data,datac]*
   (Times @@ sys[time]),{x1,0,1},{x2,0,1}]]
```

■ Common Information

```
phi[x_] := Times @@ x

phi3[x_] := phiround[x,phi,{0,3/8,5/8,1}]

phi4[x_] := phiround[x,phi,{0,1}]
```

```
test2={{{0, 0}, 0}, {{1, 1}, 1},
  {{0.4921976512515784, 0.6567022718937255},
   0.3232273157976671},
  {{0.4318405796090456, 0.916524025621182},
   0.395792266449867},
  {{0.1817978201605658, 0.01905256469642804},
   0.00346371473027877},
  {{0.0875176290939841, 0.4669830661251967},
   0.0408692507743164},
  {{0.5301873141694764, 0.822623711486269},
   0.4361446561650311},
  {{0.5282877576882771, 0.983386652896693},
   0.5195111297993741},
  {{0.0813181165792199, 0.4985288836434664},
   0.04053942987822774},
  {{0.05026429468732048, 0.5640697638801792},
   0.02835256883588061},
  {{0.2926663140648067, 0.3993713978357307},
   0.1168825549474928},
  {{0.2748510578765808, 0.5112077448767638},
   0.1405059894740797},
  {{0.1254410738042156, 0.1499798419835248},
   0.01881363242739993},
  {{0.949182608156626, 0.3951977675322207},
   0.3751148477239091},
  {{0.6332434225526374, 0.4932775700897993},
   0.3123647767521129},
  {{0.5173420285475804, 0.4786737419110386},
```

■ Systems

```
sysa[t_] := {TruncatedPDF[x1,
               NormalDistribution[E^(-3/10 t) // N,
             -(E^(-3/10 t)-1/2)^2+1/4+1/10 // N]],
             TruncatedPDF[x2,
               NormalDistribution[E^(-5/10 t) // N,
             -(E^(-5/10 t)-1/2)^2+1/4+1/10 // N]]};

sysb[t_] := {TriangularPDF[x1,4/5],
             TriangularPDF[x2,4/5]};

sysc[t_] := {PDF[BetaDistribution[25/16,1],x1],
             PDF[BetaDistribution[25/16,1],x2]};
```

■ Expected System States - A

contavg[1,sysa,phi]

0.361647

sdcontavg[1,sysa,test2]

0.368992

discrete[sysa,x1,x2,{0,1/4,1/2,3/4,1},1,phi3]

0.404406

discrete[sysa,x1,x2,{0,1/2,1},1,phi4]

0.448679

■ Expected System States - B

contavg[1,sysb,phi]

0.36

sdcontavg[1,sysb,test2]

0.365522

discrete[sysb,x1,x2,{0,1/4,1/2,3/4,1},1,phi3]

0.411163

discrete[sysb,x1,x2,{0,1/2,1},1,phi4]

0.472656

■ Expected System States - C

contavg[1,sysc,phi]

0.371802

sdcontavg[1,sysc,test2]

0.380515

discrete[sysc,x1,x2,{0,1/4,1/2,3/4,1},1,phi3]

0.411276

discrete[sysc,x1,x2,{0,1/2,1},1,phi4]

0.437498

Continuous Bounds Calculations

Note: Some groups of cells have been "closed" in this notebook for space. Feel free to open them when reading this notebook in *Mathematica*.

■ Examples 1 and 2 (n=2)

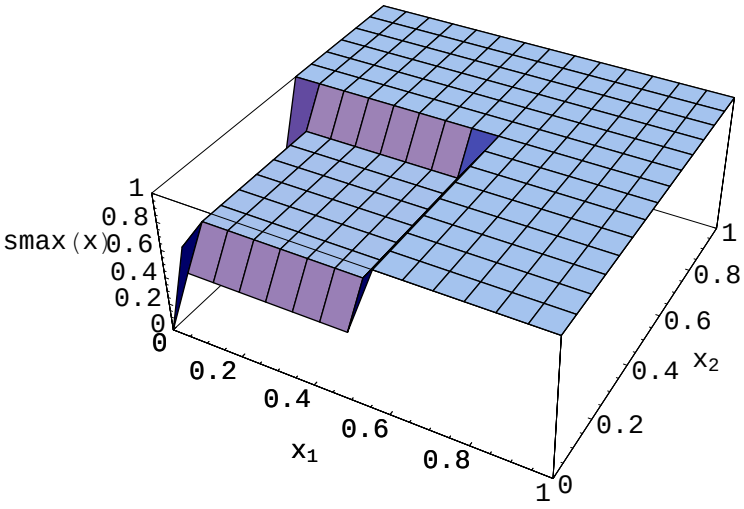
```
f={1,1};

inp={{{0,0},0},{{0,1/2},1/2},{{1/2,0},1/2},
     {{1/2,1/2},3/4},{{1,1},1}};

f2  = {TruncatedPDF[x,WeibullDistribution[3/2,2]],
       TruncatedPDF[x,WeibullDistribution[3,4]]}

f2c  = {NonTruncatedCDF[x,WeibullDistribution[3/2,2]],
        NonTruncatedCDF[x,WeibullDistribution[3,4]]}

f2in  =TruncatedPDF[x1,WeibullDistribution[3/2,2]]*
       TruncatedPDF[x2,WeibullDistribution[3,4]]
```

■ Data Sets

■ Pictures (Example 1/2, n=2)

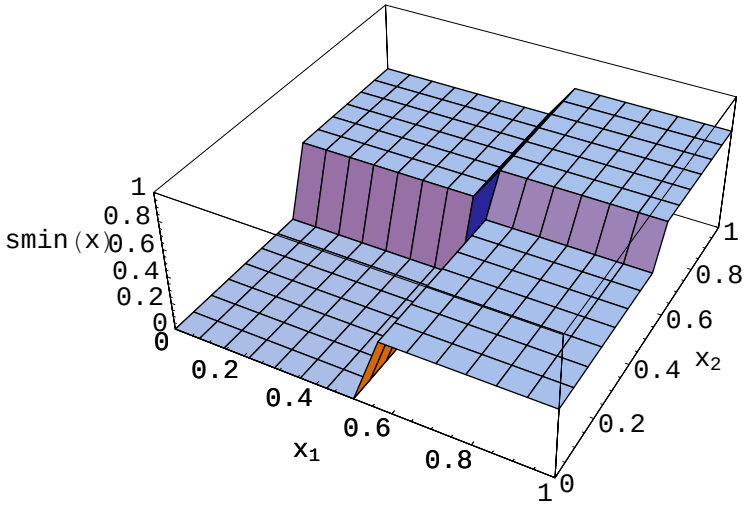
■ Example 1

```
Plot3D[PhiMax[{x1,x2},inp],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smax(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```




```
- SurfaceGraphics -
```

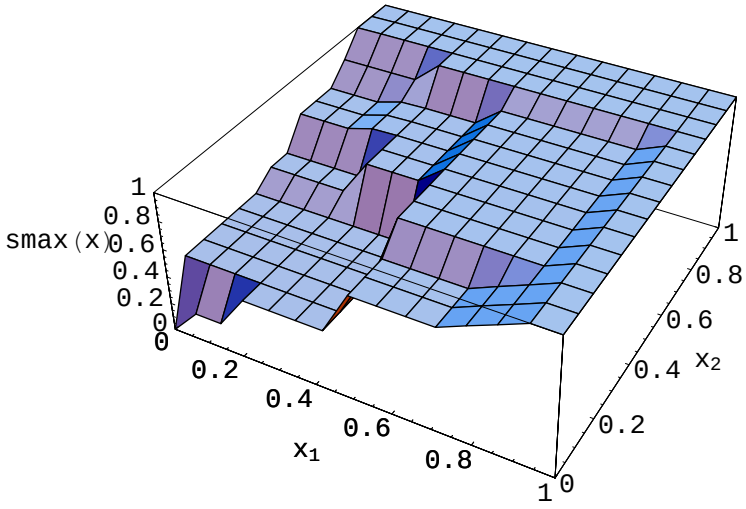
```
Plot3D[PhiMin[{x1,x2},inp],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smin(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```




```
- SurfaceGraphics -
```

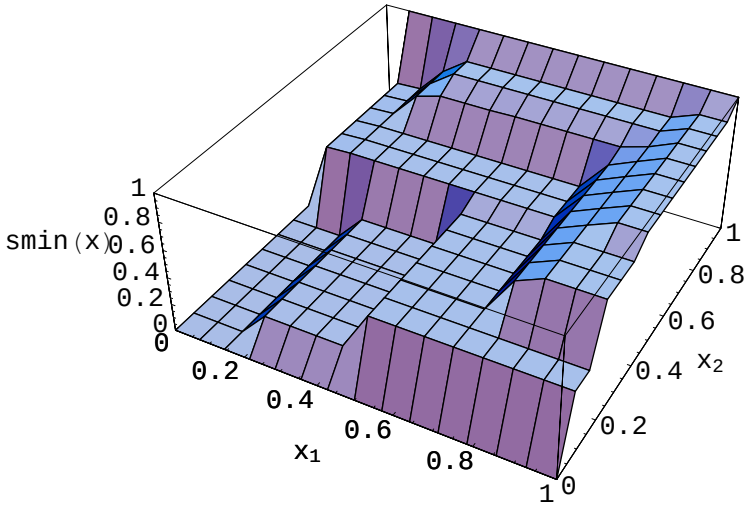
■ Example 2

```
Plot3D[PhiMax[{x1,x2},test2],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smax(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



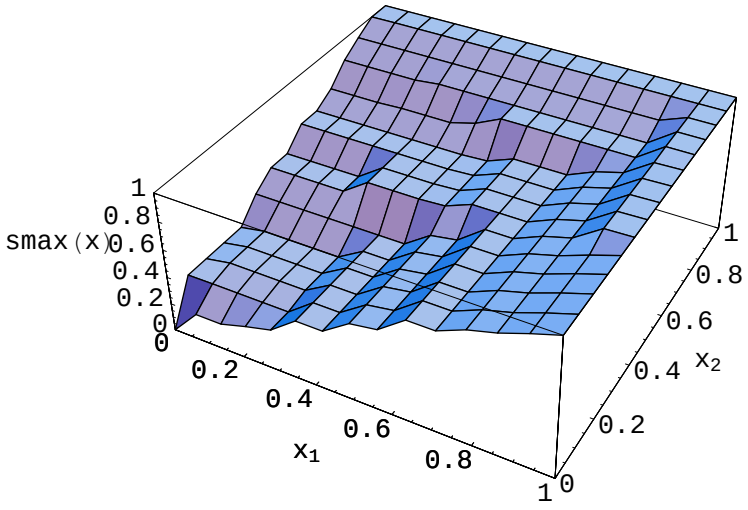

```
- SurfaceGraphics -
```

```
Plot3D[PhiMin[{x1,x2},test2],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smin(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



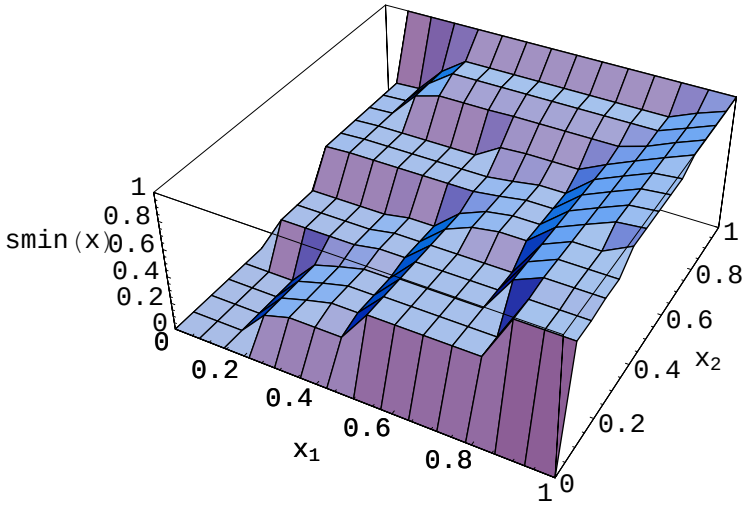

```
- SurfaceGraphics -
```

```
Plot3D[PhiMax[{x1,x2},test2b],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smax(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



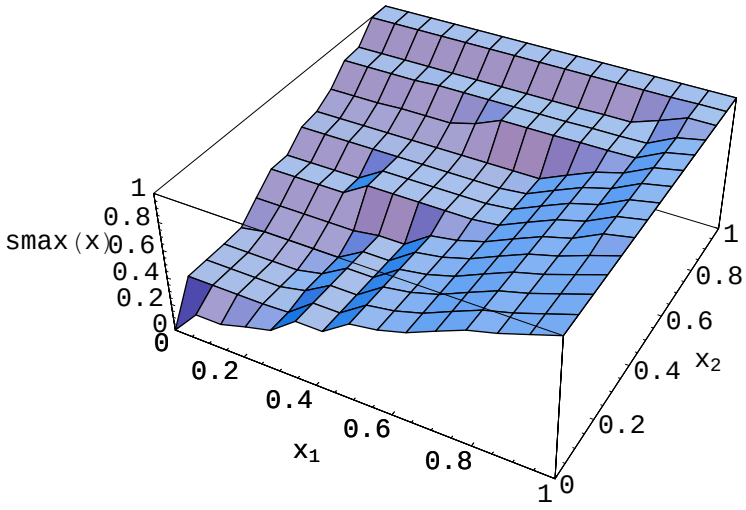

```
- SurfaceGraphics -
```

```
Plot3D[PhiMin[{x1,x2},test2b],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smin(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



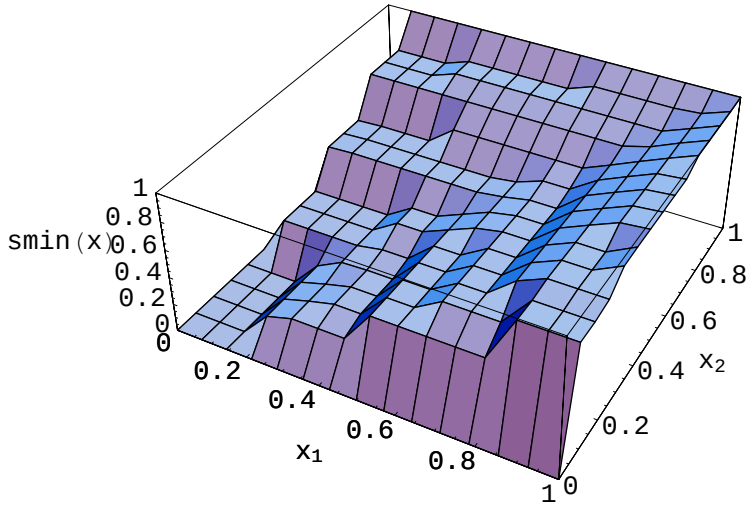

```
- SurfaceGraphics -
```

```
Plot3D[PhiMax[{x1,x2},test2c],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smax(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



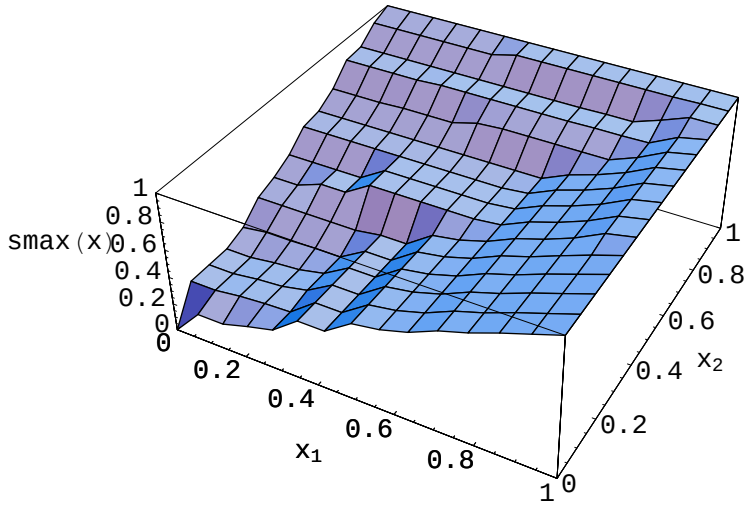

```
- SurfaceGraphics -
```

```
Plot3D[PhiMin[{x1,x2},test2c],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smin(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



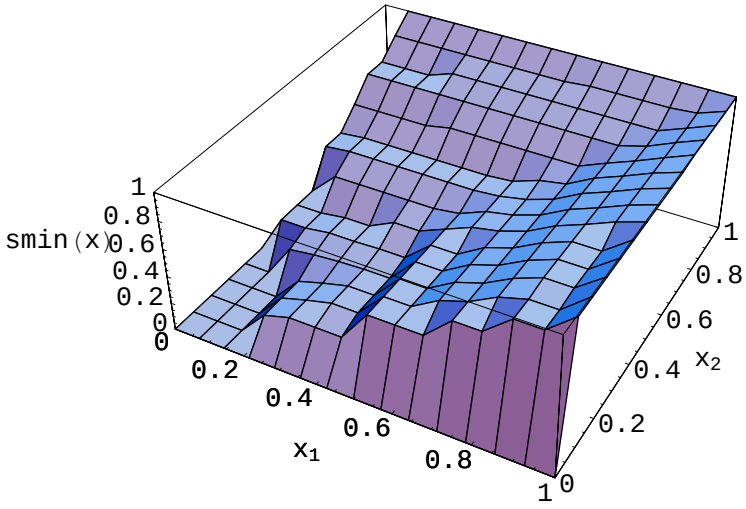

```
- SurfaceGraphics -
```

```
Plot3D[PhiMax[{x1,x2},test2d],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smax(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



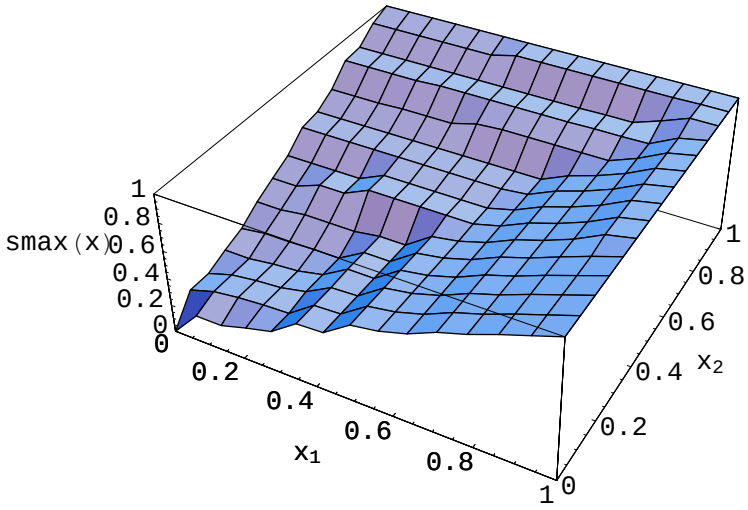

```
- SurfaceGraphics -
```

```
Plot3D[PhiMin[{x1,x2},test2d],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smin(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



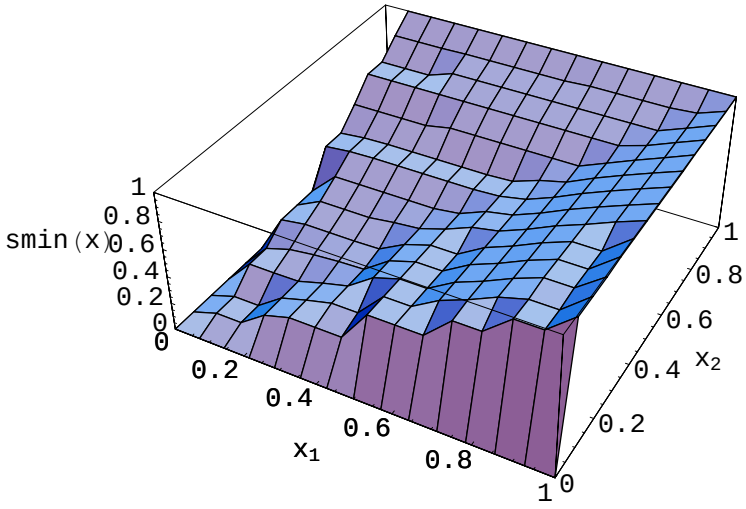

```
- SurfaceGraphics -
```

```
Plot3D[PhiMax[{x1,x2},test2e],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smax(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



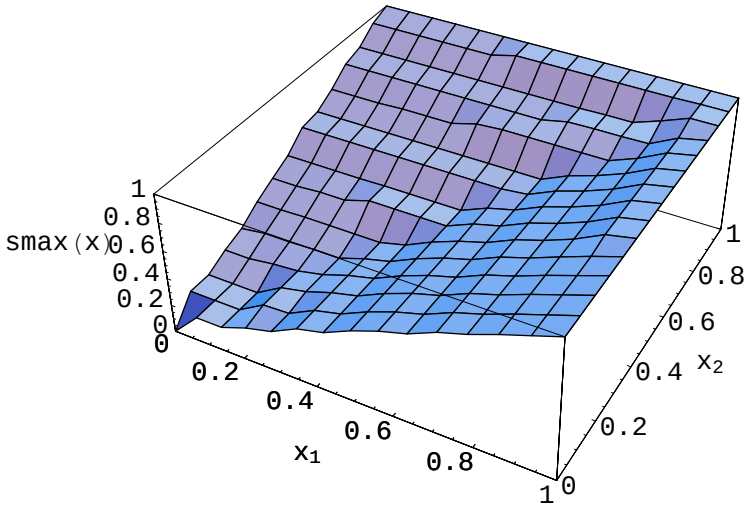

```
- SurfaceGraphics -
```

```
Plot3D[PhiMin[{x1,x2},test2e],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smin(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



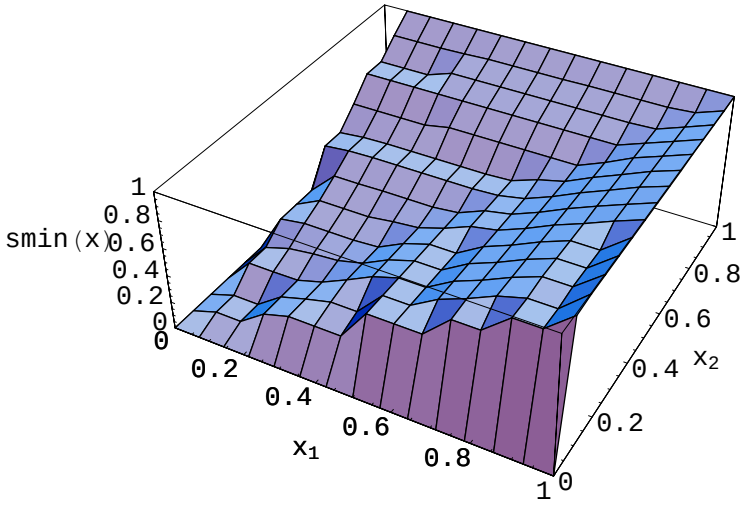

```
- SurfaceGraphics -
```

```
Plot3D[PhiMax[{x1,x2},test2f],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smax(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



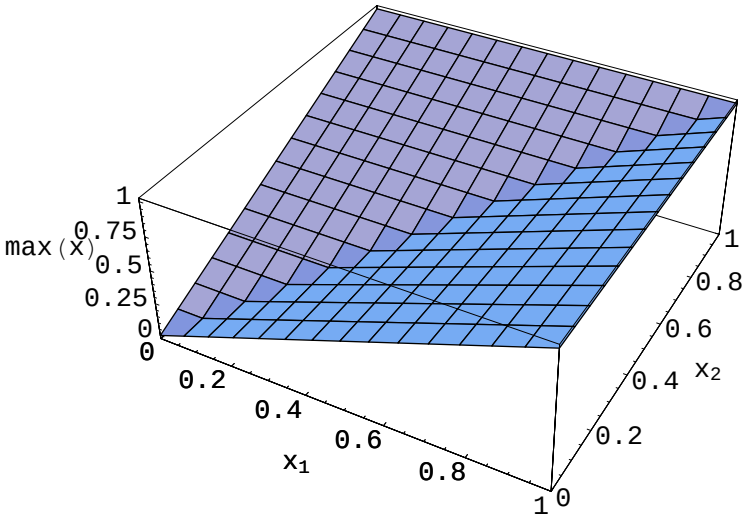

```
- SurfaceGraphics -
```

```
Plot3D[PhiMin[{x1,x2},test2f],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"smin(x)"},
 PlotRange->{{0,1},{0,1},{0,1}}]
```



- SurfaceGraphics -

```
Plot3D[Max[{x1,x2}],{x1,0,1},{x2,0,1},
 AxesLabel->{Subscript[x,1],Subscript[x,2],"max(x)"}]
```



- SurfaceGraphics -

```
NIntegrate[Max[{x1,x2}],{x1,0,1},{x2,0,1}]
```

```
0.666667
```

■ Misc Verifications (Examples 1 and 2)

■ Example 1

```
Bounds2[inp,f]
```

```
{0.4375 , 0.9375 }
```

■ Example 2

```
Bounds2[test2,f]
```

```
{0.503475 , 0.805656 }
```

```
NIntegrate[PhiMin[{x1,x2},test2],
   {x1,0,1},{x2,0,1}]
```

```
0.503117
```

```
NIntegrate[PhiMax[{x1,x2},test2],
   {x1,0,1},{x2,0,1}]
```

```
0.805417
```

```
Bounds2[test2,f2c,0]
```

```
{0.704411 , 0.927995 }
```

```
Bounds2[test2b,f2c,0]
```

```
{0.725887 , 0.866069 }
```

```
Bounds2[test2c,f2c,0]
```

```
{0.753227 , 0.851401 }
```

```
Bounds2[test2d,f2c,0]
```

```
{0.776228 , 0.846707 }
```

```
Bounds2[test2e,f2c,0]
```

```
{0.782112 , 0.844489 }
```

```
Bounds2[test2f,f2c,0]
```

```
{0.783856 , 0.84045 }
```

```
NIntegrate[f2in*PhiMax[{x1,x2},test2],
   {x1,0,1},{x2,0,1}]
```

```
0.927716
```

```
NIntegrate[f2in*PhiMin[{x1,x2},test2],
   {x1,0,1},{x2,0,1}]
```

```
0.703564
```

```
NIntegrate[f2in*Max[{x1,x2}],{x1,0,1},{x2,0,1}]
```

```
0.810897
```

■ Example 2 RMS Error Calculations

```
phi2[x_] := Max[x]
```

```
PhiMinRMSError[test2,phi2,10]
```

```
0.298629
```

```
PhiMaxRMSError[test2,phi2,10]
```

```
0.164497
```

■ Example 3 (n=4)

```
sysi = {TruncatedPDF[x,WeibullDistribution[3/2,2]],
       TruncatedPDF[x,WeibullDistribution[3,4]],
       PDF[BetaDistribution[25/16,1],x],
       TruncatedPDF[x,NormalDistribution[0.75,0.2]]}
```

```
sysic = {NonTruncatedCDF[x,WeibullDistribution[3/2,2]],
       NonTruncatedCDF[x,WeibullDistribution[3,4]],
       CDF[BetaDistribution[25/16,1],x],
       NonTruncatedCDF[x,NormalDistribution[0.75,0.2]]}
```

```
sysin =TruncatedPDF[x1,WeibullDistribution[3/2,2]]*
       TruncatedPDF[x2,WeibullDistribution[3,4]]*
       PDF[BetaDistribution[25/16,1],x3]*
       TruncatedPDF[x4,NormalDistribution[0.75,0.2]]
```

■ Data Set "data"...

■ Misc Verifications (Example 3)

Bounds2[data[[1,{1,2,3}]],{1,1,1,1}]

{0.0141986 , 0.980771 }

1-(Times @@ data[[1,3,1]])*(1-data[[1,3,2]])

0.980771

Bounds2[data[[1,{1,2,3}]],sysi]

{0.0690306 , 0.999928 }

Bounds2[data[[1]],sysic,0]

{0.377211 , 0.973481 }

Bounds2[data[[2]],sysic,0]

{0.285189, 0.968605}

Bounds2[data[[3]],sysic,0]

{0.190256, 0.951458}

Discrete Bounds Calculations

■ Discrete Bounds Example

First, we define the system:

```
phi[x_] := Select[systab,#[[1]]==x &,1][[1,2]]
```

```
fphi = Range[0,5]
```

```
{0, 1, 2, 3, 4, 5}
```

```
p = {Range[0,5], Range[0,4]}
```

```
{{0, 1, 2, 3, 4, 5}, {0, 1, 2, 3, 4}}
```

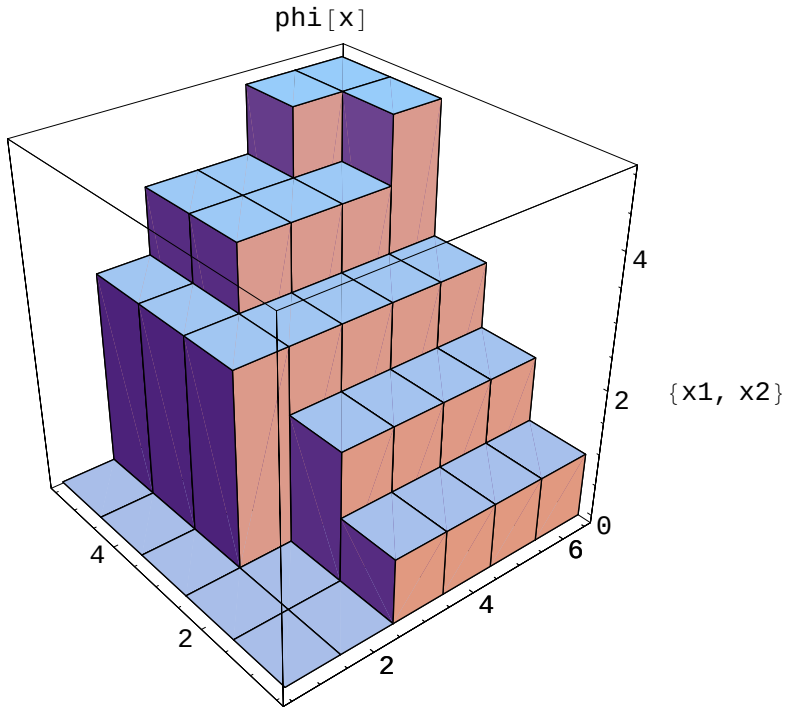
```
systab =    {{{0,0},0},
             {{1,0},0},
             {{2,0},1},
             {{3,0},1},
             {{4,0},1},
             {{5,0},1},
             {{0,1},0},
             {{1,1},0},
             {{2,1},2},
             {{3,1},2},
             {{4,1},2},
             {{5,1},2},
             {{0,2},0},
             {{1,2},3},
             {{2,2},3},
             {{3,2},3},
             {{4,2},3},
             {{5,2},3},
             {{0,3},0},
             {{1,3},3},
             {{2,3},4},
             {{3,3},4},
             {{4,3},4},
             {{5,3},5},
             {{0,4},0},
             {{1,4},3},
```

We now assume that we have knowledgeof only SOME of the structure function values:

```
systab2 =   {{{0,0},0},
             {{2,0},1},
             {{5,1},2},
             {{5,2},3},
             {{0,3},0},
             {{1,3},3},
             {{3,3},4},
             {{5,3},5},
             {{0,4},0},
             {{5,4},5}};
```

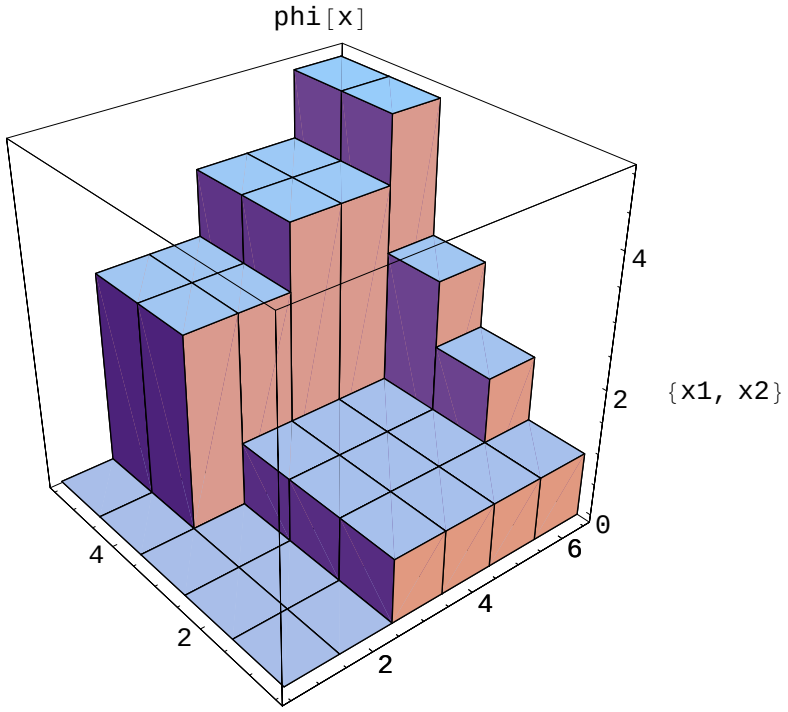
Now we compare the structure function graph of the "true" structure function to the upper and lower bounds on it created from knowledge of the "reduced set" only:

```
<<Graphics‘Graphics3D‘

BarChart3D[Table[phi[{x1,x2}],{x1,0,5},{x2,0,4}],
   ViewPoint->{-1.819,-2.233, 1.775},
   AxesLabel->{"x1","x2"},
   PlotLabel->"phi[x]"]
```



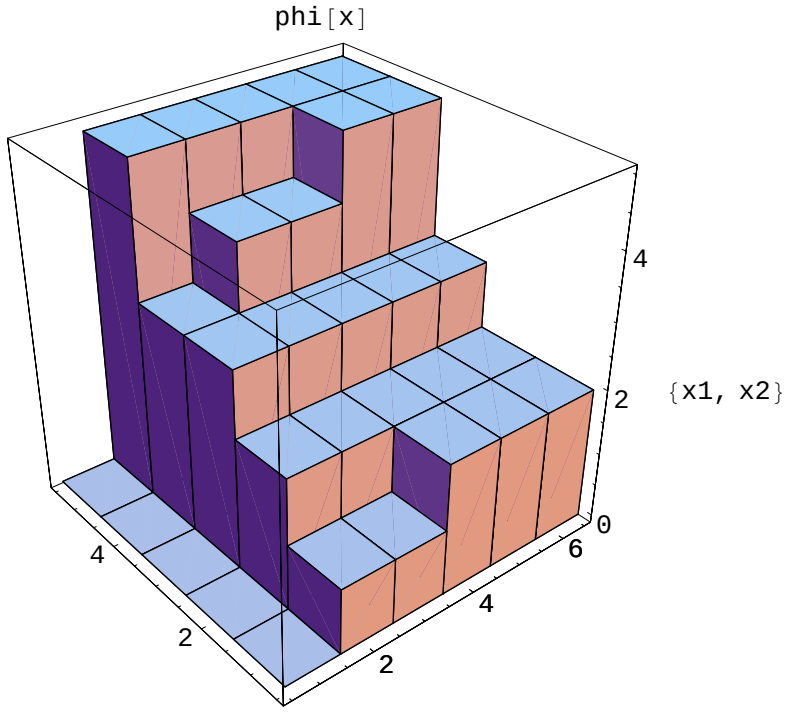

```
- Graphics3D -
```

```
BarChart3D[Table[PhiMin[{x1,x2},systab2],
              {x1,0,5},{x2,0,4}],
   ViewPoint->{-1.819,-2.233, 1.775},
   AxesLabel->{"x1","x2"},
   PlotLabel->"phi[x]"]
```



- Graphics3D -

```
BarChart3D[Table[PhiMax[{x1,x2},systab2],
              {x1,0,5},{x2,0,4}],
   ViewPoint->{-1.819,-2.233, 1.775},
   AxesLabel->{"x1","x2"},
   PlotLabel->"phi[x]"]
```




```
- Graphics3D -
```

Now let's assume that each of the states of each component has the following probability of occurring:

```
pprob = {{0.1,0.2,0.3,0.2,0.15,0.05},
         {0.2,0.2,0.3,0.1,0.2}};
```

Here are the probabilities of the SYSTEM being in any of its possible states (from 0 to 5), for both the full set and our bounds on the reduced set:

```
anse=SystemFromDirectEnumeration[p,phi,fphi,pprob]
```

```
{0.18, 0.14, 0.14, 0.33, 0.165, 0.045}
```

```
ansl=SystemFromDirectEnumerationLow[p,
      systab2,fphi,pprob]
```

```
{0.24, 0.465, 0.01, 0.165, 0.105, 0.015}
```

```
ansh=SystemFromDirectEnumerationHigh[p,
      systab2,fphi,pprob]
```

```
{0.1, 0.1, 0.26, 0.29, 0.05, 0.2}
```

Hence, we have the following limits on the reliability (here, expected state) of the system based on the reduced set:

```
{ansl,ansh} . fphi
```

```
{1.475, 2.69}
```

This may be compared to the "true" value, based on complete knowledge of the entire structure function:

```
anse . fphi
```

```
2.295
```

■ Reduced Boundary Point Sets

We now turn to the question of what should be done if knowledge of the boundary point sets (paths and cuts, in the binary case) is not complete.

First, let's consider a system that in reality is a series arrangement of four binary components. The four minimal cuts for such a system is as follows:

```
cuts={{1},{2},{3},{4}};
```

We now write this in "boundary point" form, for consistency with multistate calculations:

```
ubps=UBPFromCuts[cuts,4]
```

```
{{{0, 1, 1, 1}, 0, Upper, Real}, {{1, 0, 1, 1}, 0, Upper, Real},
 {{1, 1, 0, 1}, 0, Upper, Real}, {{1, 1, 1, 0}, 0, Upper, Real}}
```

Here is a table of the structure function value for all possible values of the components:

```
sysarray[ubps_List] :=
{#,SystemStateFromUBP[ubps,{0,1},#]}& /@
(Apply[List,Flatten[Array[Unique[],
   Table[2,{Length[ubps[[1,1]]]}]]],2]-1)
```

ubps

```
{{{0, 1, 1, 1}, 0, Upper, Real}, {{1, 0, 1, 1}, 0, Upper, Real},
 {{1, 1, 0, 1}, 0, Upper, Real}, {{1, 1, 1, 0}, 0, Upper, Real}}
```

sysarray[ubps]

```
{{{0, 0, 0, 0}, 0}, {{0, 0, 0, 1}, 0}, {{0, 0, 1, 0}, 0},
 {{0, 0, 1, 1}, 0}, {{0, 1, 0, 0}, 0}, {{0, 1, 0, 1}, 0},
 {{0, 1, 1, 0}, 0}, {{0, 1, 1, 1}, 0}, {{1, 0, 0, 0}, 0},
 {{1, 0, 0, 1}, 0}, {{1, 0, 1, 0}, 0}, {{1, 0, 1, 1}, 0},
 {{1, 1, 0, 0}, 0}, {{1, 1, 0, 1}, 0}, {{1, 1, 1, 0}, 0},
 {{1, 1, 1, 1}, 1}}
```

Now let's consider the case where one of the paths (corresponding to the minimal cut consisting of component 2) is missing:

sysarray[Drop[ubps,{2}]]

```
{{{0, 0, 0, 0}, 0}, {{0, 0, 0, 1}, 0}, {{0, 0, 1, 0}, 0},
 {{0, 0, 1, 1}, 0}, {{0, 1, 0, 0}, 0}, {{0, 1, 0, 1}, 0},
 {{0, 1, 1, 0}, 0}, {{0, 1, 1, 1}, 0}, {{1, 0, 0, 0}, 0},
 {{1, 0, 0, 1}, 0}, {{1, 0, 1, 0}, 0}, {{1, 0, 1, 1}, 1},
 {{1, 1, 0, 0}, 0}, {{1, 1, 0, 1}, 0}, {{1, 1, 1, 0}, 0},
 {{1, 1, 1, 1}, 1}}
```

Note that this structure function is still coherent, and if we did not know in advance that a cut was absent we would have no way of telling this based upon the results we obtained from our reduced cut set. This suggests that, unlike the scattered-data case where it is CLEAR what is missing and that our input data is incomplete, in cases where boundary points are missing strong assumptions would have to be made about what is missing in order to take that fact into account. Attempting this may be of questionable value.

One exception to this may be a multistate case where boundary points are not given for any level below some level k. In this case, we may reasonably take this to mean that levels of performance for the system below level k are not of interest to the customer, and to set the system state associated with any component less than (in the vector-comparison sense) any of the lower boundary points of level k to zero.

Terje Aven Example

■ Introduction

This example follows one given in T. Aven, "On performance measures for multistate monotone systems", *Reliability Engineering and System Safety,* vol 41, 1993, pp 259-266.

■ System Definition

First, we define the system under consideration, using the given information:

```
p = { {0,1}, {0,1}, {0,1,2} };

phi[x_] := Min[ x[[1]] + x[[2]], x[[3]] ]

fphi = {0,1,2};

pprob = {{0.04,0.96}, {0.04,0.96}, {0.01,0.02,0.97}};
```

■ System Distribution Calculation

Now, using the given information, we calculate the distribution of the entire system. The answer provided below is in the form of {P[phi=0], P[phi=1], P[phi=2]}.

```
ans = SystemFromDirectEnumeration[p, phi, fphi, pprob]
```

```
{0.011584 , 0.094464 , 0.893952 }
```

To check our work against the text, we request this answer in the form of {P[phi>=0], P[phi>=1], P[phi>=2]}.

```
PToQ[ans]
```

```
{1, 0.988416, 0.893952}
```

As a further check, we calculate the same system based on the given minimal paths (lower boundary points), and check that the same results are turned as when the structure function was used directly.

```
lbps = {{{1,1,2}, 2, "Lower", "Real"},
        {{0,1,1}, 1, "Lower", "Real"},
        {{1,0,1}, 1, "Lower", "Real"}};

SystemFromLBPInclusionExclusion[p,
   lbps,fphi,pprob]==ans
```

```
True
```

Now, we assemble the system matrix which is utilized by succeeding reliability measures

```
sys = SystemMatrix[fphi, ans]
```

```
{{0, 0.011584}, {1, 0.094464}, {2, 0.893952}}
```

■ Reliability Measure Calculation

■ Expected State

First, we calculate the expected state of the system, scaled by the maximal state of the system

```
maxstate = First[Last[sys]];

ExpectedState[sys] / maxstate
```

```
0.941184
```

■ Lost Output

According to the problem statement, we wish to consider a time period of one year, where the desired time unit is hours. Thus, we shall perform these computations with the target time t* = 8760.

■ Expected Value

First, we calculate the expected lost output for any time in the future t*.

```
elo = ExpectedLostOutput[sys, t, tstar]
```

```
0.117632 tstar
```

Now, we calculate this measure for a particular value of t*, that is the customer's time of interest, t*=8760.

```
elo /. tstar->8760
```

```
1030.46
```

■ Variance

We can calculate an upper bound for the variance of this result, based on the use of Schwartz's inequality.

```
voou = VarianceOfOutputUB[sys, t, 8760]
```

$$9.74281 \times 10^{6}$$

The standard deviation is thus (in hours):

```
Sqrt[voou]
```

```
3121.35
```

IEEE Paper Calculations

■ Function Definitions

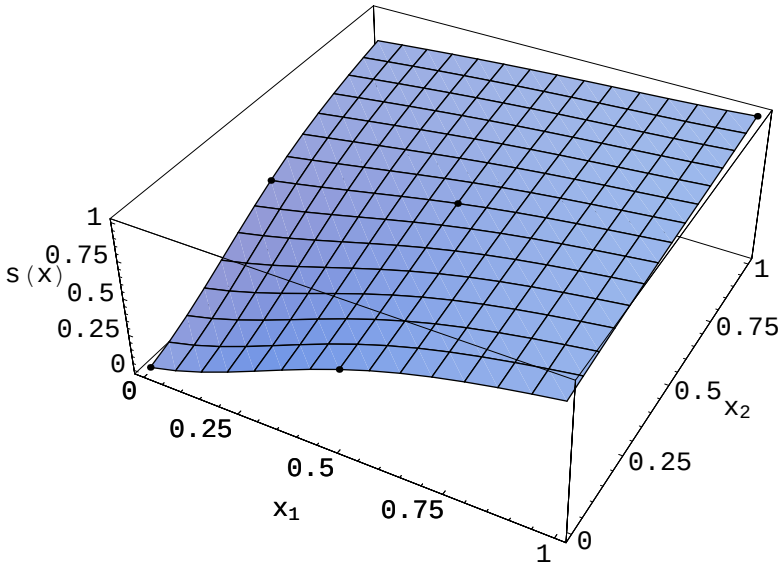
```
Ex3Calc[sys_,data_,rsq_:(1/2),ag_:2] := Module[{datac},
 datac=MultiQuadricC[data,rsq];
 NIntegrate[MultiQuadric[{x1,x2,x3,x4},data,datac,rsq]*
  (Times @@ sys), {x1,0,1},{x2,0,1},{x3,0,1},{x4,0,1},
```

■ Example 1

```
inp={{{0,0},0},{{0,1/2},1/2},{{1/2,0},1/2},
     {{1/2,1/2},3/4},{{1,1},1}};

inpc=MultiQuadricC[inp,1/6]
```

```
{2.03356, -0.80133, -0.80133, -0.241413, 0.272716}
```

```
Show[Plot3D[MultiQuadric[{x1,x2},inp,inpc,1/6],
       {x1,0,1},{x2,0,1},
           DisplayFunction->Identity],
     Graphics3D[Point /@ Flatten /@ inp,
           DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{Subscript[x,1],
                 Subscript[x,2],
                 "s(x)"}]
```

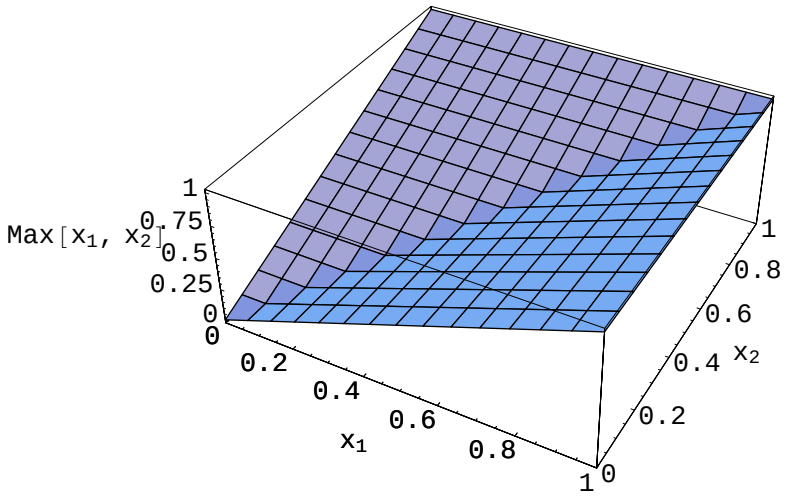



```
- Graphics3D -
```

■ Example 2

```
phi2[x_] := Max[x]
```

```
Plot3D[phi2[{x1,x2}],{x1,0,1},{x2,0,1},
   AxesLabel->{Subscript[x,1],Subscript[x,2],
      Max[Subscript[x,1],Subscript[x,2]]}]
```



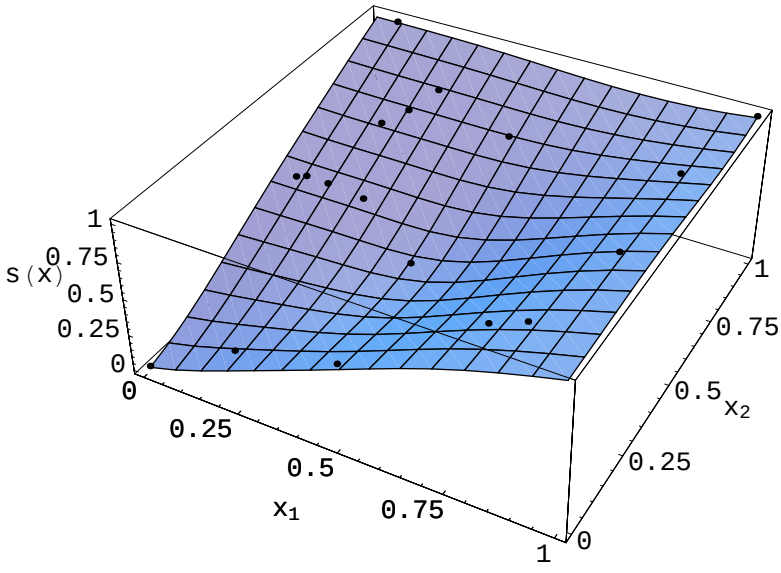

```
- SurfaceGraphics -
```

■ test2 data

```
test2={{{0, 0}, 0}, {{1, 1}, 1},
  {{0.05382814230747877, 0.5269933099816168},
   0.5269933099816168},
  {{0.919453745115708, 0.745936810438473},
   0.919453745115708},
  {{0.07700710503691687, 0.5360976784253796},
   0.5360976784253796},
  {{0.4839118816503948, 0.01681316095999332},
   0.4839118816503948},
  {{0.2600614719159675, 0.846481126833969},
   0.846481126833969},
  {{0.1739922019402756, 0.7211021773922957},
   0.7211021773922957},
  {{0.2188129353828022, 0.7736049138734915},
   0.7736049138734915},
  {{0.7529625079372377, 0.1613917491155758},
   0.7529625079372377},
  {{0.2389423683429295, 0.5249468545903034},
   0.5249468545903034},
  {{0.4831361478896971, 0.779426955843833},
   0.779426955843833},
  {{0.1907266505849009, 0.06138568074620764},
   0.1907266505849009},
  {{0.836819888375213, 0.1667087053726033},
   0.836819888375213},
  {{0.1368985082774221, 0.5343923707645909},
   0.5343923707645909},
  {{0.917366143259505, 0.4207718949341304},
   0.917366143259505},
  {{0.05989140324050526, 0.998294692339211},
   0.998294692339211},
  {{0.4334542616091103, 0.4039587339741371},
   0.4334542616091103}};

test2c=MultiQuadricC[test2,1/6]
```

```
{4.16814, 0.778698, -25.6014, 0.300392, 24.6319, -0.34898,
 -5.33281, -6.17273, 11.4533, -5.67398, -16.3528, -0.248098,
 -3.27014, 4.63301, 11.1447, -2.23432, -0.440217, 9.2096}
```

```
Show[Plot3D[MultiQuadric[{x1,x2},test2,test2c,1/6],
       {x1,0,1},{x2,0,1},
           DisplayFunction->Identity],
     Graphics3D[Point /@ ((Flatten /@ test2)+
        Table[{0,0,0.01},{Length[test2]}]),
           DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{Subscript[x,1],Subscript[x,2],
                 "s(x)"}]
```




```
- Graphics3D -
```

```
MultiQuadricRMSError[test2,phi2,10,1/6]
```

```
0.024003
```

```
ShepardRMSError[test2,phi2,10]
```

```
0.10636
```

```
theo=NIntegrate[Max[{x1,x2}],{x1,0,1},{x2,0,1}]
```

```
0.666667
```

```
ans=NIntegrate[MultiQuadric[{x1,x2},test2,test2c,1/6],
    {x1,0,1},{x2,0,1}]
```

```
0.66521
```

Abs[1-ans/theo]

0.00218459

MultiQuadricRMSError[test2,phi2,10,0.22]

0.0238958

- **test3 data**

```
test3={{{0, 0}, 0}, {{0.002290222063174745,
    0.361890337628408}, 0.361890337628408},
  {{0.05382814230747877, 0.5269933099816168},
   0.5269933099816168},
  {{0.05989140324050526, 0.998294692339211},
   0.998294692339211},
  {{0.07700710503691687, 0.5360976784253796},
   0.5360976784253796},
  {{0.1288260893700694, 0.2054324240068765},
   0.2054324240068765},
  {{0.1368985082774221, 0.534392370764591},
   0.534392370764591},
  {{0.1441739471281251, 0.3365323130588904},
   0.3365323130588904},
  {{0.1739922019402756, 0.7211021773922957},
   0.7211021773922957},
  {{0.1858624062833268, 0.5022053753849117},
   0.5022053753849117},
  {{0.1907266505849009, 0.06138568074620764},
   0.1907266505849009},
  {{0.2188129353828022, 0.7736049138734915},
   0.7736049138734915},
  {{0.2389423683429295, 0.5249468545903035},
   0.5249468545903035},
  {{0.2491451016095178, 0.1190804460287405},
   0.2491451016095178},
  {{0.2600614719159675, 0.846481126833969},
   0.846481126833969},
  {{0.2754760526305477, 0.7214151103017512},
   0.7214151103017512},
  {{0.347855773201758, 0.1698734038050016},
   0.347855773201758},
  {{0.4116506311730054, 0.782468090275657},
   0.782468090275657},
  {{0.4334542616091104, 0.4039587339741371},
   0.4334542616091104},
  {{0.4350075078928447, 0.6212858214136522},
   0.6212858214136522},
  {{0.4435519748103141, 0.5750494383147284},
   0.5750494383147284},
  {{0.4831361478896972, 0.7794269558438331},
   0.7794269558438331},
  {{0.4839118816503948, 0.01681316095999332},
   0.4839118816503948},
  {{0.4858841198676993, 0.975341978053681},
   0.975341978053681},
  {{0.5652062120708216, 0.07435512660557503},
   0.5652062120708216},
  {{0.6940323014408909, 0.2797875506124515},
   0.6940323014408909},
```

```
  {{0.6963225235040657, 0.6416778882408596},
   0.6963225235040657},
  {{0.7472294513062484, 0.933371550073869},
   0.933371550073869},
  {{0.7529625079372378, 0.1613917491155758},
   0.7529625079372378},
  {{0.836819888375213, 0.1667087053726033},
   0.836819888375213},
  {{0.838006633081569, 0.3323319715799101},
   0.838006633081569},
  {{0.888781950309773, 0.1640393654989242},
   0.888781950309773},
  {{0.917366143259505, 0.4207718949341304},
   0.917366143259505},
  {{0.919453745115708, 0.7459368104384731},
   0.919453745115708},
  {{0.936205142028753, 0.3874053135293446},
   0.936205142028753}, {{1, 1}, 1}};

MultiQuadricRMSError[test3,phi2,10,1/6]

0.0153861
```

■ Example 3

■ Define Distributions and Structure Functions

```
sysi = {TruncatedPDF[x1,WeibullDistribution[3/2,2]],
       TruncatedPDF[x2,WeibullDistribution[3,4]],
       PDF[BetaDistribution[25/16,1],x3],
       TruncatedPDF[x4,NormalDistribution[3/4,1/5]]};

phirawa[x_] := (Log[x[[1]]+1]/Log[2]/4)+
                  ((E^x[[2]]-1)/(E-1)/4)+
                  ((x[[3]]^3)/4)+
                  (Sqrt[x[[4]]]/4) // N

phirawb[x_] := ((x[[1]]^(5/6))/4)+
                  ((x[[2]]^(2))/4)+
                  ((x[[3]]^(3))/4)+
                  ((x[[4]]^(3/2))/4) // N

phirawc[x_] := (x[[1]]^(1/3))*
                  (x[[2]]^(4/3))*
                  (x[[3]]^(1/4))*
                  (x[[4]]^(1/5)) // N
```

Find Theoretical Results

```
{

NIntegrate[(Log[x1+1]/Log[2]/4) sysi[[1]],{x1,0,1}]+
NIntegrate[((E^x2-1)/(E-1)/4) sysi[[2]],{x2,0,1}]+
NIntegrate[((x3^3)/4) sysi[[3]],{x3,0,1}]+
NIntegrate[(Sqrt[x4]/4) sysi[[4]],{x4,0,1}],

NIntegrate[((x1^(5/6))/4) sysi[[1]],{x1,0,1}]+
NIntegrate[((x2^(2))/4) sysi[[2]],{x2,0,1}]+
NIntegrate[((x3^(3))/4) sysi[[3]],{x3,0,1}]+
NIntegrate[((x4^(3/2))/4) sysi[[4]],{x4,0,1}],

NIntegrate[(x1^(1/3)) sysi[[1]],{x1,0,1}]*
NIntegrate[(x2^(4/3)) sysi[[2]],{x2,0,1}]*
NIntegrate[(x3^(1/4)) sysi[[3]],{x3,0,1}]*
NIntegrate[(x4^(1/5)) sysi[[4]],{x4,0,1}]

}
```

```
{0.620369, 0.542374, 0.444722}
```

data

```
(* Data below was produced with:
n=20;

datasites=ExtremaAdd[{#,0}& /@ RandomGenerate[n-2,4]];

data={{#[[1]],phirawa[#[[1]]]}& /@ datasites,
      {#[[1]],phirawb[#[[1]]]}& /@ datasites,
      {#[[1]],phirawc[#[[1]]]}& /@ datasites};
*)

data = {{{{0, 0, 0, 0}, 0}, {{1, 1, 1, 1}, 1.},
     {{0.5236573143475254, 0.82631728719576,
       0.5847735478816752, 0.1475133296028847},
      0.4848393960576952},
     {{0.7542332254244829, 0.836248729294307,
       0.4896071937147578, 0.1408316021503213},
      0.5161324083853068},
     {{0.7095328663615365, 0.5052520218728464,
       0.03608599719979527, 0.4951690349791916},
      0.4649812409828609},
     {{0.8517075323667031, 0.4163584719878832,
       0.3069451692950627, 0.1196398595058461},
      0.3910537523113055},
     {{0.655104475602236, 0.3225573080724096,
       0.7336720470871906, 0.7443192124240596},
      0.5515276254308965},
```

```
 {{0.08307452145984999, 0.3529998609974765,
   0.139341639182566, 0.5210415799360321},
  0.2715097077502861},
 {{0.5594172071123244, 0.5266825738017164,
   0.554568091300891, 0.3735282503331474},
  0.4565548233382118},
 {{0.805183981687842, 0.6904338445074102,
   0.06496089758613312, 0.232696648182826},
  0.4784073030269244},
 {{0.09565111532630488, 0.1851818226345636,
   0.02887490038633783, 0.7375276132036344},
  0.2772506406722665},
 {{0.2439435829596022, 0.7688233506466804,
   0.7219297310912749, 0.6178877536977881},
  0.5376788586073749},
 {{0.5888391073573663, 0.446266042574271,
   0.988257684004084, 0.8735685412737289},
  0.7237874378123696},
 {{0.5057645858975162, 0.09326618157679439,
   0.8489160448215178, 0.3525269613376964},
  0.4632262643042111},
 {{0.9463473787851921, 0.5665836077750778,
   0.2943479535206274, 0.978998711004549},
  0.6048297819899635},
 {{0.1411633970973502, 0.8761497632676679,
   0.2293870559344944, 0.7463020628217231},
  0.4705455140162648},
 {{0.04551228177104538, 0.6909679406331043,
   0.2005121555481566, 0.008774449618088858},
  0.1863466482288775},
 {{0.8015686988114429, 0.9221445899864239,
   0.4785824244568814, 0.3908866959203006},
  0.6163962778780862},
 {{0.2127295914540769, 0.4758785474121528,
   0.4903247404527972, 0.5173181546465722},
  0.3675154806989664},
 {{0.7069650055565609, 0.3826123658353584,
   0.6414086956312789, 0.1647911933088755},
  0.4281305122692342}},
{{{0, 0, 0, 0}, 0}, {{1, 1, 1, 1}, 1.},
 {{0.5236573143475254, 0.82631728719576,
   0.5847735478816752, 0.1475133296028847},
  0.380674911768668},
 {{0.7542332254244829, 0.836248729294307,
   0.4896071937147578, 0.1408316021503213},
  0.4150160920218085},
 {{0.7095328663615365, 0.5052520218728464,
   0.03608599719979527, 0.4951690349791916},
  0.3387658138468705},
 {{0.8517075323667031, 0.4163584719878832,
   0.3069451692950627, 0.1196398595058461},
  0.2796138690126095},
 {{0.655104475602236, 0.3225573080724096,
   0.7336720470871906, 0.7443192124240596},
```

```
  0.4610163652696073},
 {{0.08307452145984999, 0.3529998609974765,
   0.139341639182566, 0.5210415799360321},
  0.1572957417571815},
 {{0.5594172071123244, 0.5266825738017164,
   0.554568091300891, 0.3735282503331474},
  0.3231303066294616},
 {{0.805183981687842, 0.6904338445074102,
   0.06496089758613312, 0.232696648182826},
  0.3560041584495659},
 {{0.09565111532630488, 0.1851818226345636,
   0.02887490038633783, 0.7375276132036344},
  0.2022854459014712},
 {{0.2439435829596022, 0.7688233506466804,
   0.7219297310912749, 0.6178877536977881},
  0.4404127034691717},
 {{0.5888391073573663, 0.446266042574271,
   0.988257684004084, 0.8735685412737289},
  0.655998843036941},
 {{0.5057645858975162, 0.09326618157679439,
   0.8489160448215178, 0.3525269613376964},
  0.3491010919018954},
 {{0.9463473787851921, 0.5665836077750778,
   0.2943479535206274, 0.978998711004549},
  0.567567207680777},
 {{0.1411633970973502, 0.8761497632676679,
   0.2293870559344944, 0.7463020628217231},
  0.4050148608067256},
 {{0.04551228177104538, 0.6909679406331043,
   0.2005121555481566, 0.008774449618088858},
  0.1406221144458399},
 {{0.8015686988114429, 0.9221445899864239,
   0.4785824244568814, 0.3908866959203006},
  0.5090052315365456},
 {{0.2127295914540769, 0.4758785474121528,
   0.4903247404527972, 0.5173181546465722},
  0.2479389025490081},
 {{0.7069650055565609, 0.3826123658353584,
   0.6414086956312789, 0.1647911933088755},
  0.3065488839475258}},
{{{0, 0, 0, 0}, 0}, {{1, 1, 1, 1}, 1.},
 {{0.5236573143475254, 0.82631728719576,
   0.5847735478816752, 0.1475133296028847},
  0.3727274337170384},
 {{0.7542332254244829, 0.836248729294307,
   0.4896071937147578, 0.1408316021503213},
  0.4053384657889699},
 {{0.7095328663615365, 0.5052520218728464,
   0.03608599719979527, 0.4951690349791916},
  0.1359211246632165},
 {{0.8517075323667031, 0.4163584719878832,
   0.3069451692950627, 0.1196398595058461},
  0.1434587636359673},
 {{0.655104475602236, 0.3225573080724096,
```

```
  0.7336720470871906, 0.7443192124240596},
 0.1676126178724005},
{{0.08307452145984999, 0.3529998609974765,
  0.139341639182566, 0.5210415799360321},
 0.0583781104379917},
{{0.5594172071123244, 0.5266825738017164,
  0.554568091300891, 0.3735282503331474},
 0.2483689393160614},
{{0.805183981687842, 0.6904338445074102,
  0.06496089758613312, 0.232696648182826},
 0.2141158585399884},
{{0.09565111532630488, 0.1851818226345636,
  0.02887490038633783, 0.7375276132036344},
 0.01872320816661444},
{{0.2439435829596022, 0.7688233506466804,
  0.7219297310912749, 0.6178877536977881},
 0.3684151965440207},
{{0.5888391073573663, 0.446266042574271,
  0.988257684004084, 0.8735685412737289},
 0.2773964834195918},
{{0.5057645858975162, 0.09326618157679439,
  0.8489160448215178, 0.3525269613376964},
 0.02625848832773189},
{{0.9463473787851921, 0.5665836077750778,
  0.2943479535206274, 0.978998711004549},
 0.3376037129564266},
{{0.1411633970973502, 0.8761497632676679,
  0.2293870559344944, 0.7463020628217231},
 0.2849292180661173},
{{0.04551228177104538, 0.6909679406331043,
  0.2005121555481566, 0.008774449618088858},
 0.05660210117184252},
{{0.8015686988114429, 0.9221445899864239,
  0.4785824244568814, 0.3908866959203006},
 0.5747050028162732},
{{0.2127295914540769, 0.4758785474121528,
  0.4903247404527972, 0.5173181546465722},
 0.1626709366852991},
{{0.7069650055565609, 0.3826123658353584,
  0.6414086956312789, 0.1647911933088755},
 0.1544004229556137}}};
```

■ Perform Reliability Assessment Based on Data (rsq=1/6)

```
{Ex3Calc[sysi,data[[1]],1/6,3],
 Ex3Calc[sysi,data[[2]],1/6,3],
 Ex3Calc[sysi,data[[3]],1/6,3]}
```

```
{0.62418, 0.545021, 0.44416}
```

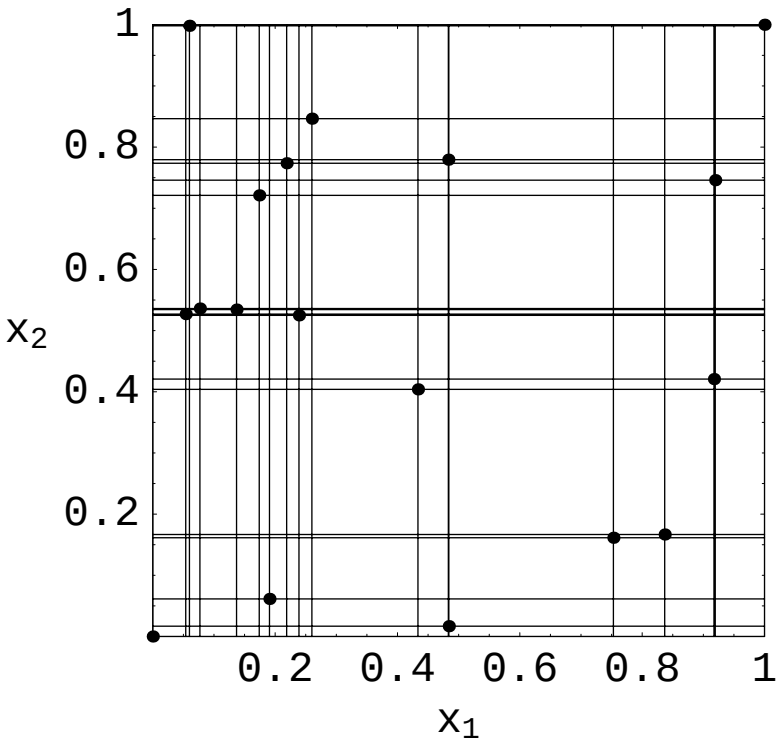
■ Perform Reliability Assessment Based on Data (rsq=1/2)

```
{Ex3Calc[sysi,data[[1]],1/2,3],
 Ex3Calc[sysi,data[[2]],1/2,3],
 Ex3Calc[sysi,data[[3]],1/2,3]}
```

```
{0.624284, 0.54382, 0.447619}
```

■ Example 2 - Monotonicity Assured

```
Show[{Graphics[{Line /@ ({{#,0},{#,1}}& /@
        Union[#[[1,1]]& /@ test2])}],
      Graphics[{Line /@ ({{0,#},{1,#}}& /@
        Union[#[[1,2]]& /@ test2])}],
      Graphics[{PointSize[0.02],
        {Point /@ (#[[1]]& /@ test2)}}]},
      AspectRatio->1,
      PlotRange->{{0,1},{0,1}},
      Frame->True,
      FrameLabel->{Subscript[x,1],Subscript[x,2]},
      RotateLabel->False,TextStyle->{FontSize->16}
]
```

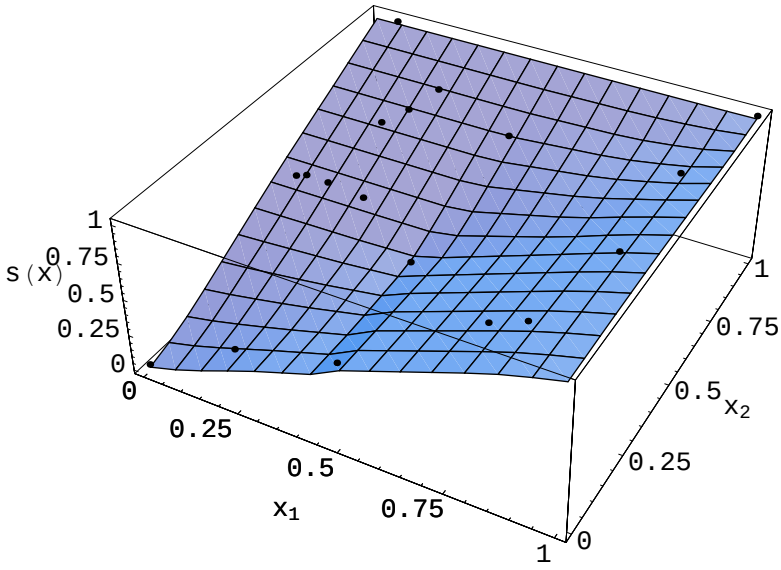


- Graphics -

```
test2mqnd2=MultiQuadricND2[test2,1/6];

MultiQuadricNDRMSError[test2mqnd2,phi2,10]
```

```
0.0228224
```

```
Show[Plot3D[MLinInt[{x1,x2},test2mqnd2],
       {x1,0,1},{x2,0,1},
           DisplayFunction->Identity],
     Graphics3D[Point /@ ((Flatten /@ test2)+
        Table[{0,0,0.02},{Length[test2]}]),
           DisplayFunction->Identity],
     DisplayFunction->$DisplayFunction,
     AxesLabel->{Subscript[x,1],
                 Subscript[x,2],
                 "s(x)"}]
```

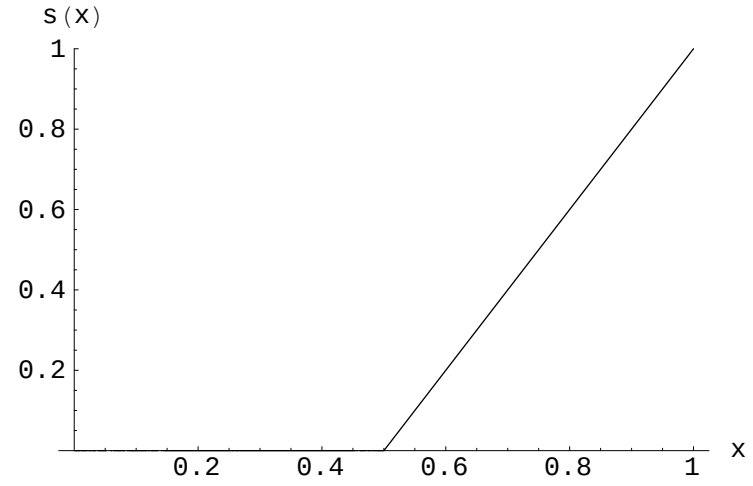


- Graphics3D -

Analysis of Effect of R^2

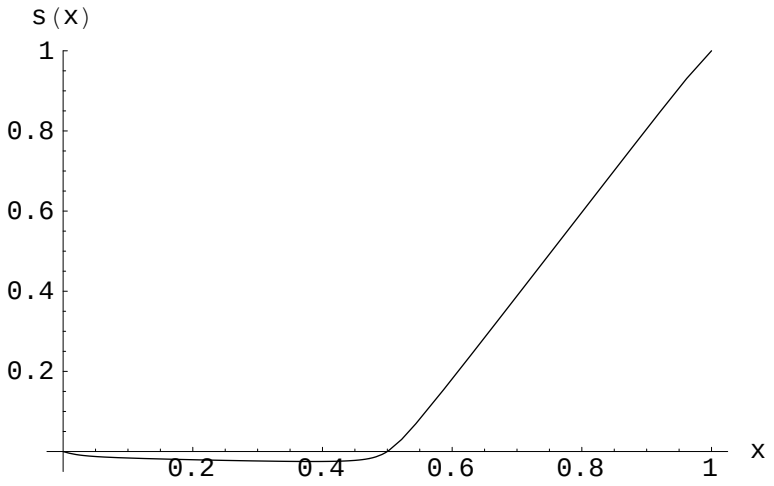
■ Calculations

```
data = {{{0}, 0}, {{1 / 2}, 0}, {{1}, 1}};

Plot[MultiQuadric[{x}, data, MultiQuadricC[data, 0], 0],
 {x, 0, 1}, PlotRange -> {0, 1},
 AxesLabel -> {TraditionalForm[x], TraditionalForm[s[x]]}]
```



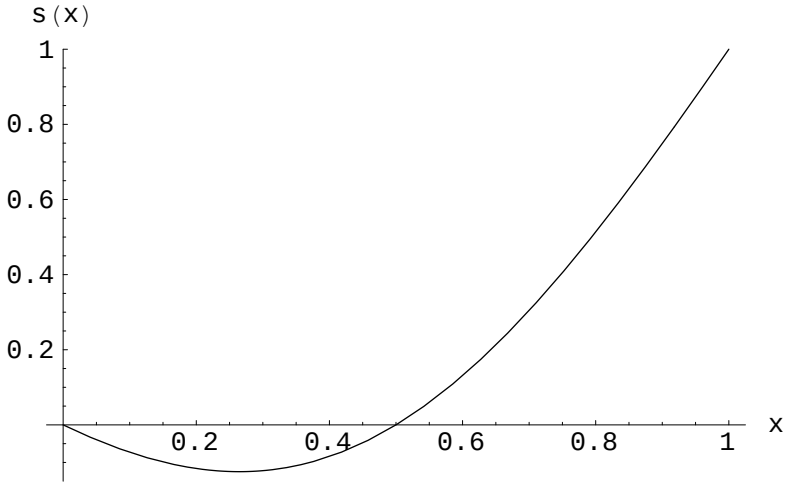

```
- Graphics -
```

```
Plot[MultiQuadric[{x}, data, MultiQuadricC[data, 1 / 1000],
 1 / 1000], {x, 0, 1}, PlotRange -> {-0.05, 1},
 AxesLabel -> {TraditionalForm[x], TraditionalForm[s[x]]}]
```




```
- Graphics -
```

```
Plot[MultiQuadric[{x}, data, MultiQuadricC[data, 1], 1],
 {x, 0, 1}, PlotRange -> {-0.15, 1},
 AxesLabel -> {TraditionalForm[x], TraditionalForm[s[x]]}]
```




```
- Graphics -
```

IIE Paper Example 1

■ Comments

In this notebook, functions are given in full rather than called through RelPack; it is hence completely self-containedIn fact, attempting to evaluate this notebook after RelPack has been loaded will result in errors, as this will cause slightly different functions with the same names to be loaded over existing "protected" ones. This notebook contains the examples in a paper which the author co-authored and has submitted to IIE Transactions

■ Functions

```
Off[General::spell]; Off[General::spell1];
```

■ Multistate Markov Models

```
PDPErlangian[M_Integer, val_, t_] := Module[{ans = Table[0, {M + 1}]},
  Do[ans[[i + 1]] = (valt)^(M-i) E^(-val t) / (M - i)!, {i, 1, M}]; ans[[1]] = 1 - Sum[ans[[i + 1]], {i, 1, M}];
  ans]
```

```
PDPErlangianSystem[m_] :=
 Transpose[{Table[i/m, {i, 0, m}], PDPErlangian[m, λ, t]}] // Simplify
```

```
PDPAdjacent[mymu_List, t_] :=
 Module[{m = Length[mymu], mu = Prepend[mymu, 0]},
  Table[pti[i, m, mu, t], {i, 0, m}]]
```

```
pti[i_Integer, m_Integer, mu_List, t_] :=
 (Sum[aij[i, j, m, mu] E^(-mu[[j+1]] t), {j, i, m}]) Product[mu[[j + 1]], {j, i+1, m}]
```

```
aij[i_Integer, j_Integer, m_Integer, mu_List] :=
 Product[If[k =!= j, 1/(mu[[k + 1]] - mu[[j + 1]]), 1], {k, m}]
```

```
SYSF[sys_List, s_] := Plus @@ (#1[[2]] &) /@ Select[sys, #1[[1]] ≤ s &]
```

```
SYSE2[sys_List] := Plus@@(#1[[1]]^2 #1[[2]]&) /@ sys
```

■ Multistate Measures

```
SYSR[sys_List, s_] := 1 - SYSF[sys, s]

SYSE[sys_List] := Plus@@(#1[[1]] #1[[2]]&) /@ sys

SYSV[sys_List] := SYSE2[sys] - (SYSE[sys])^2

SYSOE[sys_List, t_, τ_, ni_: False, rules_: Null] :=
 If[ni, NIntegrate[SYSE[sys], {t, 0, τ}],
  Integrate[SYSE[sys], {t, 0, τ}, rules]]

SYSOVUB[sys_List, t_, τ_, ni_: False, rules_: Null] :=
 If[ni, NIntegrate[Sqrt[SYSE2[sys]], {t, 0, τ}]^2 -
   NIntegrate[SYSE[sys], {t, 0, τ}]^2,
  Integrate[Sqrt[SYSE2[sys]], {t, 0, τ}, rules]^2 -
   Integrate[SYSE[sys], {t, 0, τ}, rules]^2]

SYSD[sys_List, t_, s_, ni_: False, rules_: Null] :=
 If[ni, NIntegrate[SYSR[sys, s], {t, 0, ∞}],
  Integrate[SYSR[sys, s], {t, 0, ∞}, rules]]

SYSTOE[sys_List, t_, ni_: False, rules_: Null] :=
 SYSOE[sys, t, ∞, ni, rules]

SYSLWE[sys_List, t_, U_, ni_: False, rules_: Null] :=
 If[ni, NIntegrate[U * SYSE[sys], {t, 0, ∞}],
  Integrate[U * SYSE[sys], {t, 0, ∞}, rules]]
```

■ Example

```
(sys = PDPErlangianSystem[3]) // MatrixForm
```

$$\begin{pmatrix} 0 & \frac{1}{2} E^{-t\lambda} (-2 + 2 E^{t\lambda} - 2 t\lambda - t^2 \lambda^2) \\ \frac{1}{3} & \frac{1}{2} E^{-t\lambda} t^2 \lambda^2 \\ \frac{2}{3} & E^{-t\lambda} t\lambda \\ 1 & E^{-t\lambda} \end{pmatrix}$$

```
{#[[1]], SYSR[sys, #[[1]]]}& /@ sys // Simplify // MatrixForm
```

$$\begin{pmatrix} 0 & \frac{1}{2} E^{-t\,\lambda} \left(2 + 2\,t\,\lambda + t^2\,\lambda^2\right) \\ \frac{1}{3} & E^{-t\,\lambda} \left(1 + t\,\lambda\right) \\ \frac{2}{3} & E^{-t\,\lambda} \\ 1 & 0 \end{pmatrix}$$

```
SYSE[sys] // Simplify
```

$$\frac{1}{6} E^{-t\,\lambda} \left(6 + 4\,t\,\lambda + t^2\,\lambda^2\right)$$

```
SYSV[sys] // Simplify
```

$$\frac{1}{36} E^{-2\,t\,\lambda} \left(36 \left(-1 + E^{t\,\lambda}\right) + 16 \left(-3 + E^{t\,\lambda}\right) t\,\lambda + 2 \left(-14 + E^{t\,\lambda}\right) t^2\,\lambda^2 - 8\,t^3\,\lambda^3 - t^4\,\lambda^4\right)$$

```
Last[FindMinimum[-(SYSV[sys] /. λ -> 3), {t, 1}]]
```

$$\{t \to 0.599214\}$$

```
SYSOE[sys, t, τ, False, Assumptions → (τ > 0)] // Simplify
```

$$\frac{12 - E^{-\lambda\,\tau} \left(12 + 6\,\lambda\,\tau + \lambda^2\,\tau^2\right)}{6\,\lambda}$$

```
SYSOVUB[sys /. λ → 3, t, 2, True]
```

```
0.338449
```

```
{#[[1]], SYSD[sys, t, #[[1]], False, Assumptions → (Re[λ] > 0)]}& /@
   Drop[sys, -1] // Simplify //
 MatrixForm
```

$$\begin{pmatrix} 0 & \frac{3}{\lambda} \\ \frac{1}{3} & \frac{2}{\lambda} \\ \frac{2}{3} & \frac{1}{\lambda} \end{pmatrix}$$

```
SYSTOE[sys, t, False, Assumptions → (Re[λ] > 0)] // Simplify
```

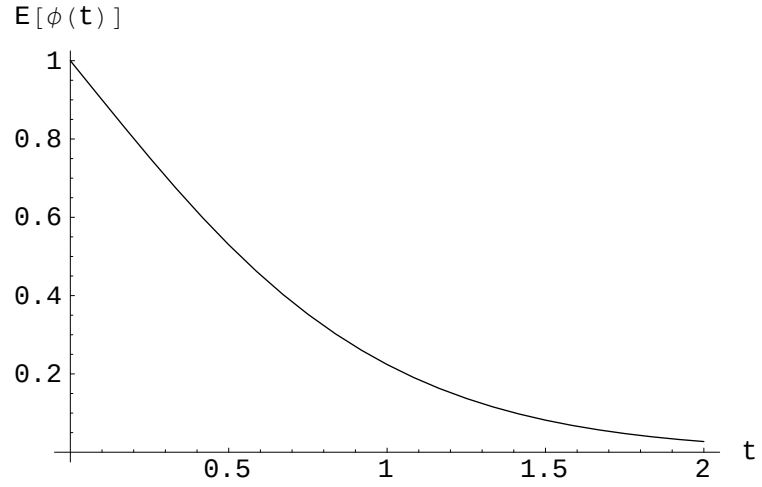
$$\frac{2}{\lambda}$$

```
U₃[t_] := E^(-t ν) ν
```

```
SYSLWE[sys, t, U₃[t], False, Assumptions → (Re[λ + ν] > 0)]
```

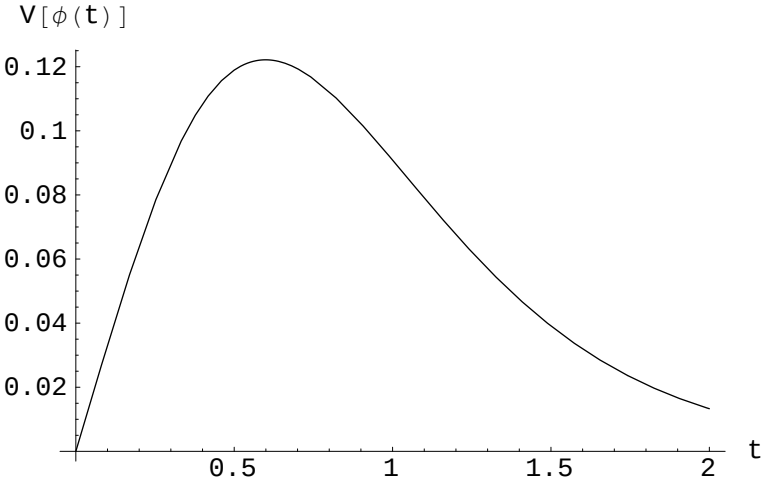
$$\frac{\nu \left(6\,\lambda^2 + 8\,\lambda\,\nu + 3\,\nu^2\right)}{3 \left(\lambda + \nu\right)^3}$$

```
Plot[SYSE[sys] /. λ → 3, {t, 0, 2}, AxesLabel -> {"t", "E[ϕ(t)]"}]
```



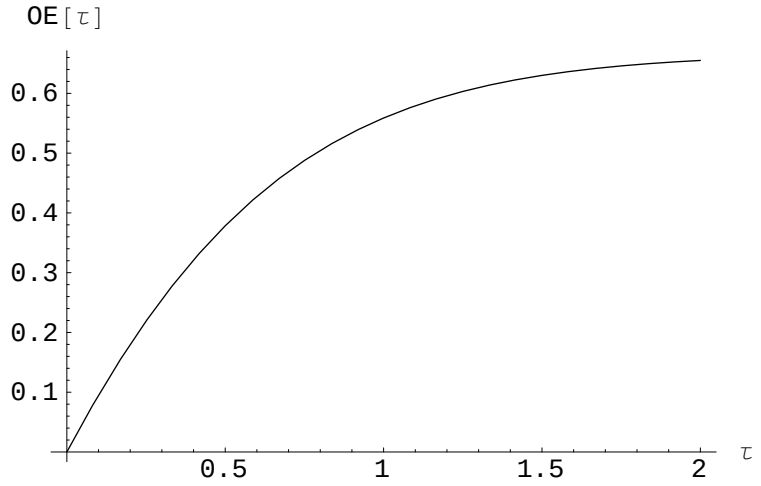
- Graphics -

```
Plot[SYSV[sys] /. λ → 3, {t, 0, 2}, AxesLabel -> {"t", "V[ϕ(t)]"}]
```



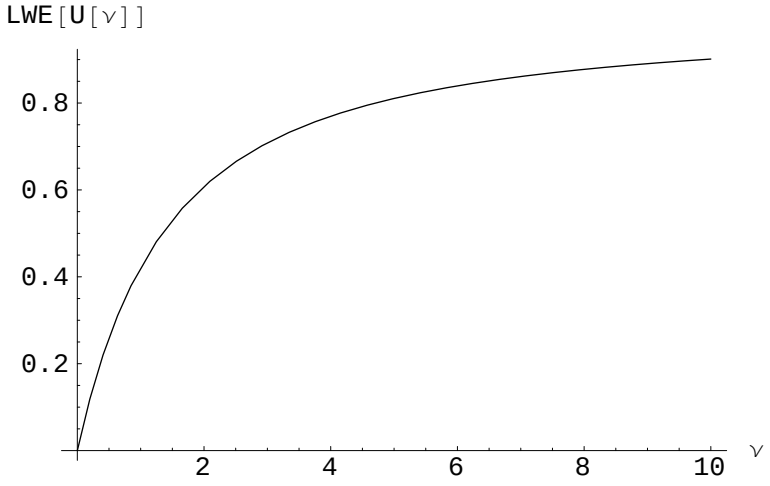
- Graphics -

```
Plot[SYSOE[sys, t, τ, False, Assumptions → (τ > 0)] /. λ → 3,
 {τ, 0, 2}, AxesLabel -> {"τ", "OE[τ]"}]
```




```
- Graphics -
```

```
Plot[Evaluate[
  SYSLWE[sys, t, U3[t], False, Assumptions → (Re[λ + ν] > 0)] /. λ → 3],
 {ν, 0, 10}, AxesLabel -> {"ν", "LWE[U[ν]]"}]
```




```
- Graphics -
```

IIE Paper Example 2

■ Functions

```
Off[General::spell]; Off[General::spell1];

<< Statistics`ContinuousDistributions`

S[cdfs_List] := 1 - Times@@(1 - cdfs)

P[cdfs_List] := Times@@cdfs

AVG[e_List] := (Plus@@e) / Length[e]

AVG2[eb_List] := Module[{e, e2}, {e, e2} = Transpose[eb];
  {(Plus@@e) / Length[e], ((Plus@@e2) + 2 Sum[e[[i]] e[[j]],
        {i, 1, Length[e] - 1}, {j, i + 1, Length[e]}]) /
    Length[e]^2}]

PROD[e_List] := Times@@e

PROD2[eb_List] :=
 Module[{e, e2}, {e, e2} = Transpose[eb]; {Times@@e, Times@@e2}]

COPROD[e_List] := 1 - Times@@(1 - e)

COPROD2[eb_List] := Module[{e, e2}, {e, e2} = Transpose[eb];
  {1 - Times@@(1 - e), 1 - 2 (Times@@(1 - e)) + Times@@(1 - 2 e + e2)}]

BinaryCDF[x_, p_] := Which[
    x < 0,  0,
    x >= 1,  1,
    True,  1 - p
    ]

MultistateCDF[x_, sys_] := Which[
      x < First[sys][[1]],  0,
      x >= Last[sys][[1]],  1,
      x >= First[sys][[1]] && x < Last[sys][[1]],
        Do[If[sys[[i, 1]] <= x < sys[[i + 1, 1]],
          Return[Sum[sys[[j, 2]], {j, i}]]]
        , {i, Length[sys] - 1}]]

μ[t_, k_] := E^(-Log[2] t/k)
```

```
σ[t_, k_] := 1/4 (1/2 - Abs[1/2 - μ[t, k]]) + 1/100

F[t_, k_, x_] := Which[
 x < 0,  0,
 x >= 1,  1,
 True,  (CDF[NormalDistribution[μ[t, k], σ[t, k]], x] -
      CDF[NormalDistribution[μ[t, k], σ[t, k]], 0]) /
    (CDF[NormalDistribution[μ[t, k], σ[t, k]], 1] -
     CDF[NormalDistribution[μ[t, k], σ[t, k]], 0])]
```

■ Continuum Measures (CDF)

Note: for repairable systems CDFR would more properly be called CDFA. Note also that t is the first argument in these expressions rather than the second.

```
CDFMoment[t_, n_, a_: 2] :=
 NIntegrate[n x^(n-1) (1 - CDFF[t, x]), {x, 0, 1}, AccuracyGoal → a]

CDFR[t_, x_] := 1 - CDFF[t, x]

CDFE[t_, a_: 2] := CDFMoment[t, 1, a]

CDFV[t_, a_: 2] := CDFMoment[t, 2, a] - CDFMoment[t, 1, a]^2

CDFOE[tau_, a_: 2] :=
 NIntegrate[CDFE[t, a], {t, 0, tau}, AccuracyGoal → a]

CDFOVUB[tau_, a_: 2] :=
 NIntegrate[Sqrt[CDFMoment[t, 2, a]], {t, 0, tau}, AccuracyGoal → a]^2 -
  CDFOE[tau, a]^2

CDFD[x_, a_: 2] := NIntegrate[CDFR[t, x], {t, 0, ∞}, AccuracyGoal → a]

CDFTOE[a_: 2] := CDFOE[∞, a]

CDFLWE[u_, a_: 2] :=
 NIntegrate[u CDFE[t, a], {t, 0, ∞}, AccuracyGoal → a]

CDFAV[tau_, s_: 0, a_: 2] :=
 NIntegrate[CDFR[t, s], {t, 0, tau}, AccuracyGoal → a] / tau

B[α_: 0.05, s_: 0, a_: 2] := FindRoot[CDFR[b, s] == 1 - α, {b, 0, 0.01}]

CDFOES[tau_, a_: 2] := CDFOE[tau, a] / tau

CDFOCT[tau_, α_: 0.05, a_: 2] :=
 {CDFOE[tau, a] + Sqrt[CDFOVUB[tau, a] / α],
  CDFOE[tau, a] - Sqrt[CDFOVUB[tau, a] / α]}
```

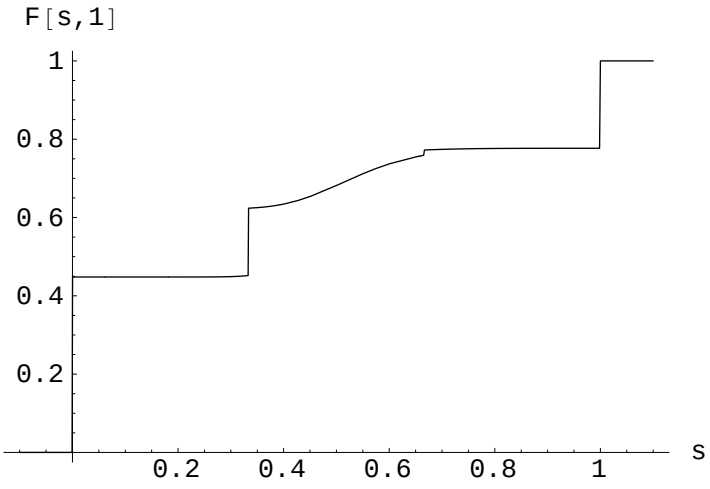
```
CDFDV[x_, a_: 2] :=
 NIntegrate[2 t CDFR[t, x], {t, 0, ∞}, AccuracyGoal → a] -
  NIntegrate[CDFR[t, x], {t, 0, ∞}, AccuracyGoal → a]^2

CDFT[α_: 0.95, a_: 2] :=
 FindRoot[CDFOE[T, a] == α CDFTOE[a], {T, 0, 0.01}]
```

■ Example A System

```
c = {MultistateCDF[x, {{0, 1/2 E^(-3 t) (-2 + 2 E^(3 t) - 6 t - 9 t^2)},
     {1/3, 9/2 E^(-3 t) t^2}, {2/3, 3 E^(-3 t) t}, {1, E^(-3 t)}}],
  BinaryCDF[x, E^(-3 t/2)], F[t, 2/3, x], F[t, 1, x], F[t, 1/2, x],
  F[t, 3/4, x]};

CDFF[t0_, x0_] :=
 P[{c[[2]], S[{c[[1]], P[{c[[3]], c[[4]], c[[5]], c[[6]]}]}]}] /.
   {t → t0, x → x0} //
  N
```

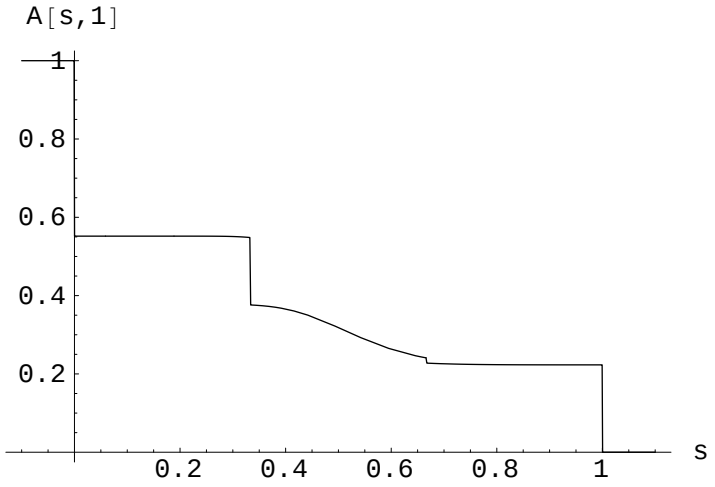
■ Example A Calculations

```
Plot[CDFF[1, s], {s, -0.1, 1.1}, AxesLabel -> {"s", "F[s,1]"}]
```



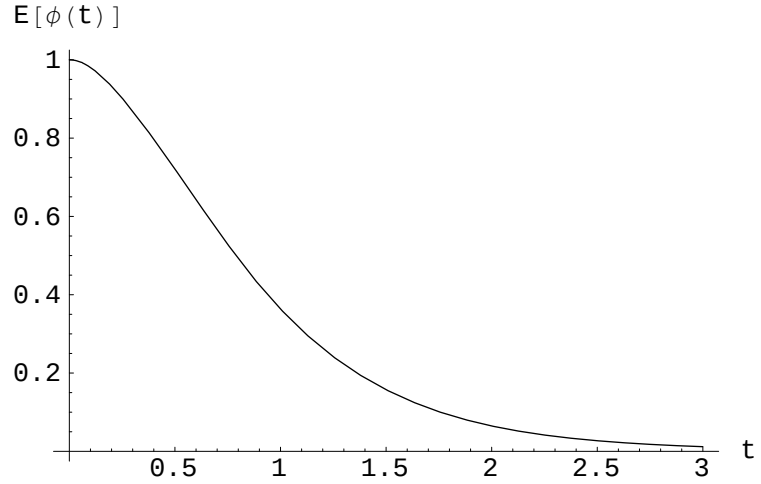

```
- Graphics -
```

```
Plot[CDFR[1, s], {s, -0.1, 1.1}, AxesLabel -> {"s", "A[s,1]"}]
```



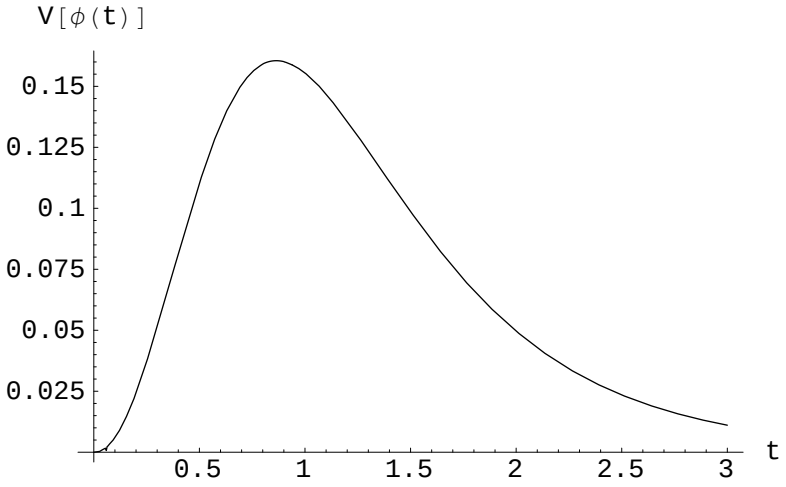

```
- Graphics -
```

```
Plot[CDFE[t, 4], {t, 0, 3}, AxesLabel -> {"t", "E[ϕ(t)]"}]
```




```
- Graphics -
```

```
Plot[CDFV[t, 4], {t, 0, 3}, AxesLabel -> {"t", "V[ϕ(t)]"}]
```




```
- Graphics -
```

```
CDFE[3, 4]
```

```
0.0119001
```

```
CDFV[3, 4]
```

```
0.0110788
```

```
CDFOE[3, 4]
```

```
0.918261
```

```
CDFOVUB[3, 4]
```

```
0.874275
```

```
CDFD[0, 4]
```

```
1.19753
```

```
CDFTOE[4]
```

```
0.925911
```

```
B[0.05, 0, 4]
```

```
{b → 0.388612 }
```

```
CDFAV[1 / 10, 0, 4]
```

```
0.999895
```

```
CDFOES[3, 4]
```

```
0.306087
```

```
CDFOCT[3, 0.05, 4]
```

```
{5.09983 , -3.26331 }
```

```
CDFDV[0, 4]
```

```
0.426002
```

```
CDFT[0.95, 4]
```

```
{T → 1.89013 }
```

■ Example B System and Calculations

```
e1e2b[tau_: 1] :=
 {{Plus@@ (#1[[1]] #1[[2]]&) /@ ({{0, 1/2 E^(-3 t) (-2 + 2 E^(3 t) - 6 t - 9 t^2)},
         {1/3, 9/2 E^(-3 t) t^2}, {2/3, 3 E^(-3 t) t}, {1, E^(-3 t)}} /. t -> tau) //
    N,
   Plus@@ (#1[[1]] ^2 #1[[2]]&) /@ ({{0, 1/2 E^(-3 t) (-2 + 2 E^(3 t) - 6 t - 9 t^2)},
         {1/3, 9/2 E^(-3 t) t^2}, {2/3, 3 E^(-3 t) t}, {1, E^(-3 t)}} /. t -> tau) //
    N},
  {Plus@@
     (#1[[1]] #1[[2]]&) /@ ({{0, 1 - E^(-3 t/2)}, {1, E^(-3 t/2)}} /. t -> tau) // N,
   Plus@@ (#1[[1]] ^2 #1[[2]]&) /@
     ({{0, 1 - E^(-3 t/2)}, {1, E^(-3 t/2)}} /. t -> tau) // N},
  {NIntegrate[(1 - (c[[3]] /. t -> tau)), {x, 0, 1}],
   NIntegrate[2 x (1 - (c[[3]] /. t -> tau)), {x, 0, 1}]},
  {NIntegrate[(1 - (c[[4]] /. t -> tau)), {x, 0, 1}],
   NIntegrate[2 x (1 - (c[[4]] /. t -> tau)), {x, 0, 1}]},
  {NIntegrate[(1 - (c[[5]] /. t -> tau)), {x, 0, 1}],
   NIntegrate[2 x (1 - (c[[5]] /. t -> tau)), {x, 0, 1}]},
  {NIntegrate[(1 - (c[[6]] /. t -> tau)), {x, 0, 1}],
   NIntegrate[2 x (1 - (c[[6]] /. t -> tau)), {x, 0, 1}]}}
```

```
eb = e1e2b[1]
```

```
{{0.224042 , 0.141063 }, {0.22313 , 0.22313 }, {0.353615 , 0.134702 },
 {0.5, 0.268168 }, {0.250076 , 0.0677752 }, {0.396909 , 0.169441 }}
```

```
eb2 = COPROD2[
  {PROD2[{AVG2[{eb[[1]], eb[[2]], eb[[3]]}], eb[[4]], eb[[5]]}],
   eb[[6]]}]
```

```
{0.417038 , 0.185319 }
```

```
{e, v} = {eb2[[1]], eb2[[2]] - eb2[[1]]^2}
```

```
{0.417038 , 0.0113983 }
```

Miscellaneous Examples

■ Tire Example

```
(tire = Transpose[{{0, 1, 2}, PDPAdjacent[{μ1, μ2}, t]}] // Simplify) //
 MatrixForm
```

$$\begin{pmatrix} 0 & \frac{\left(1-E^{-t\mu_2}\right)\mu_1+\left(-1+E^{-t\mu_1}\right)\mu_2}{\mu_1-\mu_2} \\ 1 & \frac{\left(E^{-t\mu_1}-E^{-t\mu_2}\right)\mu_2}{-\mu_1+\mu_2} \\ 2 & \end{pmatrix}$$

```
ExpectedState[tire]
```

$$2\,E^{-t\mu_2}+\frac{\left(E^{-t\mu_1}-E^{-t\mu_2}\right)\mu_2}{-\mu_1+\mu_2}$$

```
tire /. {μ1 → 1, μ2 → 2} // MatrixForm
```

$$\begin{pmatrix} 0 & -1+E^{-2t}-2\left(-1+E^{-t}\right) \\ 1 & 2\left(-E^{-2t}+E^{-t}\right) \\ 2 & E^{-2t} \end{pmatrix}$$

```
ExpectedState[tire] /. {μ1 → 1, μ2 → 2} // Simplify
```

$$2\,E^{-t}$$

■ Continuous Bounds Example

```
NIntegrate[MultiQuadric[{x1, x2}, {{{0, 0}, 0}, {{0, 1/2}, 1/2},
   {{1/2, 0}, 1/2}, {{1/2, 1/2}, 3/4}, {{1, 1}, 1}}, Evaluate[
   MultiQuadric[{{{0, 0}, 0}, {{0, 1/2}, 1/2}, {{1/2, 0}, 1/2},
  {{1/2, 1/2}, 3/4}, {{1, 1}, 1}}, 1/6], 1/6]], {x1, 0, 1},
```

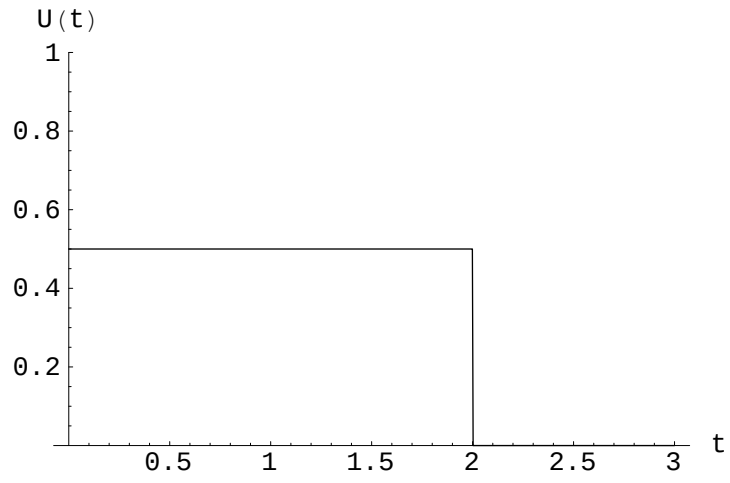
```
0.71351
```

```
Bounds2[{{{0, 0}, 0}, {{0, 1/2}, 1/2},
  {{1/2, 0}, 1/2}, {{1/2, 1/2}, 3/4}, {{1, 1}, 1}},
 {1, 1}]
```

```
{0.4375, 0.9375}
```

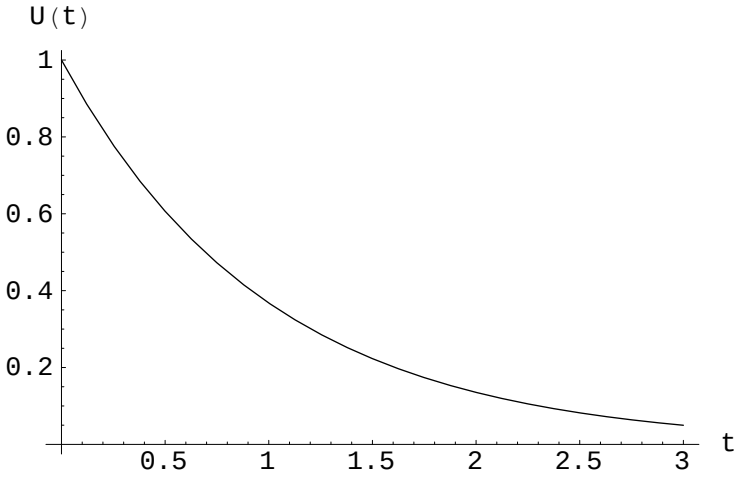
■ U(t) Illustrations

```
Plot[PDF[UniformDistribution[0, 2], t], {t, 0, 3},
 PlotRange -> {0, 1}, AxesLabel -> {"t", "U(t)"}]
```



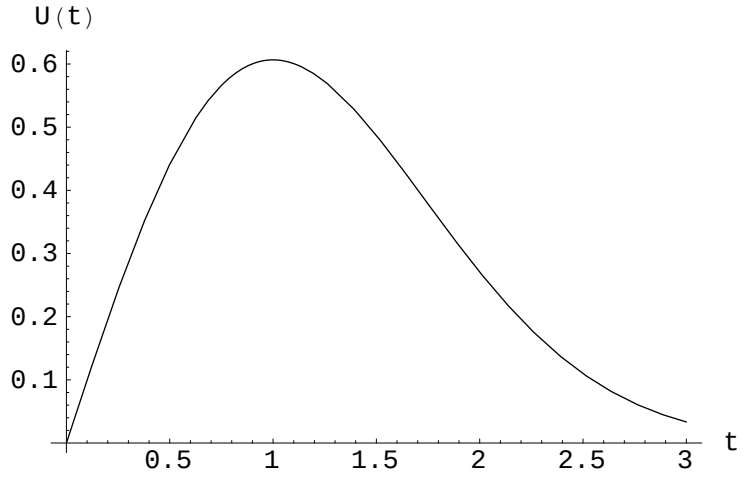
- Graphics -

```
Plot[PDF[ExponentialDistribution[1], t], {t, 0, 3},
 AxesLabel -> {"t", "U(t)"}]
```



- Graphics -

```
Plot[PDF[RayleighDistribution[1], t], {t, 0, 3},
 AxesLabel -> {"t", "U(t)"}]
```




```
- Graphics -
```

Appendix J

SOFTWARE FUNCTION CODE

These are the functions which comprise the set of original *Mathematica* packages that were documented in Appendix I. If the disk accompanying this dissertation is available, these functions may be called from it. If not, then by entering all of them and calling any standard *Mathematica* packages mentioned at the beginning of each section, the functionality utilized in Appendix I will be duplicated.

J.1 ContinuousOptimization

The functions in this section require that the Measures, Distributions, and Statistics`ContinuousDistributions packages also be loaded.

```
MonteCarlo2[cdflist_List,x_,phi_,d_:0.1,min_:0,max_:1] :=
  Module[{n,data,inp},
    n=0.9604/d^2; (* p +- d is 95% CI *)
    inp=Table[CDFRandom[cdflist[[i]],x,n,min,max],
            {i,1,Length[cdflist]}];
    data=phi[#]& /@ Transpose[inp];
    MultistateCDF[x,FnEstimate[data]]
]


ContDisc[cdflist_List,x_,phi_,n_:5,min_:0,max_:1] :=
  Module[{p,fphi,pprob},
    p=Table[min + (i-1)/(n-1) (max-min),
      {Length[cdflist]},{i,n}];
    fphi=p[[1]];
    pprob=Table[If[j==1,cdflist[[i]] /. x->p[[i,j]],
      (cdflist[[i]] /. x->p[[i,j]]) -
      (cdflist[[i]] /. x->p[[i,j-1]])],
        {i,Length[cdflist]},{j,n}];
    Transpose[{fphi,
      SystemFromDirectEnumeration2[p,phi,fphi,pprob]}]]
```

```
SystemFromDirectEnumeration2[p_List,phi_Symbol,
      fphi_List,pprob_List] :=
  Module[{ans = Table[0, {Length[fphi]}], tab2, temp},
    tab2 = systable3[p, phi, fphi];
    Do[temp=Transpose[Select[tab2,
      #[[2]]==fphi[[i]] &]][[1]];
       Do[Do[temp[[j,k]] =
             pprob[[k,Position[p[[k]],
               temp[[j,k]]][[1,1]]]]
          ,{k,Length[p]}]
       ,{j,Length[temp]}];
       ans[[i]] = Plus @@ Apply[Times, temp, 1]
    ,{i,Length[fphi]}];
    ans
  ]


systable3[p_List,phi_Symbol,fphi_List] :=
    {#, phirnd[#,phi,fphi]}& /@ perm3[p]


phirnd[x_,phi_,fphi_] := Module[{temp,i=1},
  temp=phi[x];
  While[temp>fphi[[i]] && i<Length[fphi],i++];
  fphi[[i]]]


perm3[p_List] := Module[{tab,size,m,n,a,i,j},
  n=Length[p];
  m=Table[Length[p[[i]]],{i,n}];
  size=Product[m[[i]],{i,n}];
  tab=Table[0,{size}];
  a=Table[1,{i,n}];
  Do[tab[[j]]=Table[p[[i,a[[i]]]],{i,n}];
     Do[If[Take[a,-(n-i)]==Take[m,-(n-i)],
           If[a[[i]]==m[[i]],a[[i]]=1,a[[i]]++]],
             {i,n-1}];
     If[a[[n]]==m[[n]],a[[n]]=1,a[[n]]++];
  ,{j,size}];
  tab]


PDFADist[pdf_,x_,a_List] :=
  Sqrt[Sum[NIntegrate[(x-(a[[i+1]]+a[[i]])/2)^2 pdf,
       {x,a[[i]],a[[i+1]]}], {i, 2, Length[a]-2}]+
```

```
      NIntegrate[(x-First[a])^2 pdf,
       {x,a[[1]],a[[2]]}]+
      NIntegrate[(x-Last[a])^2 pdf,
       {x,a[[-2]],a[[-1]]}]]

RD[pdf_,x_,n_,min_:0,max_:1] :=
   Switch[n,2,RD02[pdf,x,min,max],
            3,RD03[pdf,x,min,max],
            4,RD04[pdf,x,min,max],
            5,RD05[pdf,x,min,max],
            6,RD06[pdf,x,min,max],
            7,RD07[pdf,x,min,max],
            8,RD08[pdf,x,min,max],
            9,RD09[pdf,x,min,max],
           10,RD10[pdf,x,min,max],
            _,False]

RD02[pdf_,x_,min_:0,max_:1] := Module[{x1,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,max}]^2,
      {x1,{min+1/2*(max-min),min+1/2*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]

RD03[pdf_,x_,min_:0,max_:1] := Module[{x1,x2,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,x2,max}]^2,
      {x1,{min+1/3*(max-min),min+1/3*(max-min)+10*$MachineEpsilon}},
      {x2,{min+2/3*(max-min),min+2/3*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,x2,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]

RD04[pdf_,x_,min_:0,max_:1] := Module[{x1,x2,x3,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,x2,x3,max}]^2,
      {x1,{min+1/4*(max-min),min+1/4*(max-min)+10*$MachineEpsilon}},
      {x2,{min+2/4*(max-min),min+2/4*(max-min)+10*$MachineEpsilon}},
      {x3,{min+3/4*(max-min),min+3/4*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,x2,x3,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]
```

```
RD05[pdf_,x_,min_:0,max_:1] := Module[{x1,x2,x3,x4,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,x2,x3,x4,max}]^2,
      {x1,{min+1/5*(max-min),min+1/5*(max-min)+10*$MachineEpsilon}},
      {x2,{min+2/5*(max-min),min+2/5*(max-min)+10*$MachineEpsilon}},
      {x3,{min+3/5*(max-min),min+3/5*(max-min)+10*$MachineEpsilon}},
      {x4,{min+4/5*(max-min),min+4/5*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,x2,x3,x4,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]


RD06[pdf_,x_,min_:0,max_:1] := Module[{x1,x2,x3,x4,x5,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,x2,x3,x4,x5,max}]^2,
      {x1,{min+1/6*(max-min),min+1/6*(max-min)+10*$MachineEpsilon}},
      {x2,{min+2/6*(max-min),min+2/6*(max-min)+10*$MachineEpsilon}},
      {x3,{min+3/6*(max-min),min+3/6*(max-min)+10*$MachineEpsilon}},
      {x4,{min+4/6*(max-min),min+4/6*(max-min)+10*$MachineEpsilon}},
      {x5,{min+5/6*(max-min),min+5/6*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,x2,x3,x4,x5,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]


RD07[pdf_,x_,min_:0,max_:1] := Module[{x1,x2,x3,x4,x5,x6,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,x2,x3,x4,x5,x6,max}]^2,
      {x1,{min+1/7*(max-min),min+1/7*(max-min)+10*$MachineEpsilon}},
      {x2,{min+2/7*(max-min),min+2/7*(max-min)+10*$MachineEpsilon}},
      {x3,{min+3/7*(max-min),min+3/7*(max-min)+10*$MachineEpsilon}},
      {x4,{min+4/7*(max-min),min+4/7*(max-min)+10*$MachineEpsilon}},
      {x5,{min+5/7*(max-min),min+5/7*(max-min)+10*$MachineEpsilon}},
      {x6,{min+6/7*(max-min),min+6/7*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,x2,x3,x4,x5,x6,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]


RD08[pdf_,x_,min_:0,max_:1] := Module[{x1,x2,x3,x4,x5,x6,x7,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,x2,x3,x4,x5,x6,x7,max}]^2,
      {x1,{min+1/8*(max-min),min+1/8*(max-min)+10*$MachineEpsilon}},
      {x2,{min+2/8*(max-min),min+2/8*(max-min)+10*$MachineEpsilon}},
      {x3,{min+3/8*(max-min),min+3/8*(max-min)+10*$MachineEpsilon}},
      {x4,{min+4/8*(max-min),min+4/8*(max-min)+10*$MachineEpsilon}},
```

```
      {x5,{min+5/8*(max-min),min+5/8*(max-min)+10*$MachineEpsilon}},
      {x6,{min+6/8*(max-min),min+6/8*(max-min)+10*$MachineEpsilon}},
      {x7,{min+7/8*(max-min),min+7/8*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,x2,x3,x4,x5,x6,x7,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]


RD09[pdf_,x_,min_:0,max_:1] := Module[{x1,x2,x3,x4,x5,x6,x7,x8,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,x2,x3,x4,x5,x6,x7,x8,max}]^2,
      {x1,{min+1/9*(max-min),min+1/9*(max-min)+10*$MachineEpsilon}},
      {x2,{min+2/9*(max-min),min+2/9*(max-min)+10*$MachineEpsilon}},
      {x3,{min+3/9*(max-min),min+3/9*(max-min)+10*$MachineEpsilon}},
      {x4,{min+4/9*(max-min),min+4/9*(max-min)+10*$MachineEpsilon}},
      {x5,{min+5/9*(max-min),min+5/9*(max-min)+10*$MachineEpsilon}},
      {x6,{min+6/9*(max-min),min+6/9*(max-min)+10*$MachineEpsilon}},
      {x7,{min+7/9*(max-min),min+7/9*(max-min)+10*$MachineEpsilon}},
      {x8,{min+8/9*(max-min),min+8/9*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,x2,x3,x4,x5,x6,x7,x8,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]


RD10[pdf_,x_,min_:0,max_:1] := Module[{x1,x2,x3,x4,x5,x6,x7,x8,x9,ans},
   ans=FindMinimum[PDFADist[pdf,x,{min,x1,x2,x3,x4,x5,x6,x7,x8,x9,max}]^2,
      {x1,{min+1/10*(max-min),min+1/10*(max-min)+10*$MachineEpsilon}},
      {x2,{min+2/10*(max-min),min+2/10*(max-min)+10*$MachineEpsilon}},
      {x3,{min+3/10*(max-min),min+3/10*(max-min)+10*$MachineEpsilon}},
      {x4,{min+4/10*(max-min),min+4/10*(max-min)+10*$MachineEpsilon}},
      {x5,{min+5/10*(max-min),min+5/10*(max-min)+10*$MachineEpsilon}},
      {x6,{min+6/10*(max-min),min+6/10*(max-min)+10*$MachineEpsilon}},
      {x7,{min+7/10*(max-min),min+7/10*(max-min)+10*$MachineEpsilon}},
      {x8,{min+8/10*(max-min),min+8/10*(max-min)+10*$MachineEpsilon}},
      {x9,{min+9/10*(max-min),min+9/10*(max-min)+10*$MachineEpsilon}},
      MaxIterations->30000];
   ans[[2]]={min,x1,x2,x3,x4,x5,x6,x7,x8,x9,max} /. ans[[2]];
   ans[[1]]=Sqrt[ans[[1]]];
   ans]


GenAns[pdf_,x_,a_] := Module[{n, ans},
   n=Length[a];
   ans=Table[{(a[[i]]+a[[i+1]])/2,
```

```
          NIntegrate[pdf,{x,a[[i]],a[[i+1]]}]},{i,n-1}];
   ans[[1,1]]=a[[1]]; ans[[n-1,1]]=a[[n]];
   ans]


TruncatedPDF[x_,mmadist_,min_:0,max_:1] :=
    PDF[mmadist,x]/(CDF[mmadist,max]-CDF[mmadist,min])


CohInputQ[data_List] := Module[{n,ans=True},
  n=Length[data];
  Do[Do[If[(LessOrEqualQ[data[[j,1]],data[[i,1]]] &&
           data[[j,2]]>data[[i,2]]) ||
           (GreaterOrEqualQ[data[[j,1]],data[[i,1]]] &&
           data[[j,2]]<data[[i,2]]),
             ans=False; Print[j,", ",i]]
     ,{i,j+1,n}]
  ,{j,n-1}];
  ans]


ExtremaAdd[data_List] := Module[{n,ans},
   ans=data;
   n=Length[data[[1,1]]];
   If[FreeQ[Transpose[data][[1]],Table[1,{n}]],
      ans=Prepend[ans,{Table[1,{n}],1}]];
   If[FreeQ[Transpose[data][[1]],Table[0,{n}]],
      ans=Prepend[ans,{Table[0,{n}],0}]];
   ans]


Shepard[x_List,data_List] := Module[{n,d},
   n=Length[data]; d=Table[1,{n}];
   Do[d[[i]]=Chop[Plus @@ ((x-data[[i,1]])^2)],{i,n}];
   If[FreeQ[d,0],
        Sum[data[[i,2]]/d[[i]],{i,n}]/
        Sum[1/d[[i]],{i,n}],
      data[[Position[d,0][[1,1]],2]]]


MultiQuadric[x_List, data_List, c_List, rsq_:(1/6),
   prec_:$MachinePrecision] :=
   N[c . Sqrt[Plus @@ Transpose[(Table[x,{Length[data]}]-
            Transpose[data][[1]])^2] + rsq],prec]
```

```
MultiQuadricC[data_List,rsq_:(1/6),prec_:$MachinePrecision] :=
   LinearSolve[N[Table[Sqrt[Plus @@ ((data[[i,1]]-
                 data[[j,1]])^2) + rsq]
      ,{i,Length[data]},{j,Length[data]}],prec],
               N[Transpose[data][[2]],prec]]


ShepardND[data_,m_:5] :=
Module[{obt,grid,n,temp,low,high,ans,a1,a0},
   n=Length[data[[1,1]]]; obt=ExtremaAdd[data];
   grid=GridGenerate[m,n];
   grid=Transpose[Sort[Transpose[{grid,Apply[Plus,
      ((grid-1/2)^2),2]}],OrderedQ[{#1[[2]],
      #2[[2]]}]&]][[1]];
   ans=Table[1/2,{Length[grid]}];
   a1=Transpose[Select[ExtremaAdd[data],
      Chop[(1-#[[2]])]==0&]][[1]];
   a0=Transpose[Select[ExtremaAdd[data],
      Chop[#[[2]]]==0&]][[1]];
   If[Length[a1]>1 || Length[a0]>1,
     Do[Do[If[LessQ[grid[[i]],a0[[j]]],
           If[FreeQ[Transpose[obt][[1]],grid[[i]]],
              obt=Prepend[obt,{grid[[i]],0}]]]
        ,{j,Length[a0]}];
        Do[If[GreaterQ[grid[[i]],a1[[j]]],
           If[FreeQ[Transpose[obt][[1]],grid[[i]]],
              obt=Prepend[obt,{grid[[i]],1}]]]
        ,{j,Length[a1]}];
     ,{i,Length[grid]}]];
   Do[If[FreeQ[Transpose[obt][[1]],grid[[i]]],
         low=0; high=1;
         temp=Shepard[grid[[i]],obt];
         Do[If[LessQ[obt[[j,1]],grid[[i]]],
               low=Max[low,obt[[j,2]]]];
            If[GreaterQ[obt[[j,1]],grid[[i]]],
               high=Min[high,obt[[j,2]]]];
         ,{j,Length[obt]}];
         If[temp<low,temp=low]; If[temp>high,temp=high];
            obt=Prepend[obt,{grid[[i]],temp}];
            ans[[i]]=temp,ans[[i]]=Select[obt,
              #[[1]]==grid[[i]]&][[1,2]]]
   ,{i,Length[grid]}];
   Transpose[{grid,ans}]]
```

```
MultiQuadricND[data_,m_:5,rsq_:(1/6),prec_:$MachinePrecision] :=
Module[{obt,grid,n,temp,tempc,low,high,ans,a1,a0},
   n=Length[data[[1,1]]]; obt=ExtremaAdd[data];
   grid=GridGenerate[m,n];
   grid=Transpose[Sort[Transpose[{grid,Apply[Plus,
      ((grid-1/2)^2),2]}],OrderedQ[{#1[[2]],
      #2[[2]]}]&]][[1]];
   ans=Table[1/2,{Length[grid]}];
   a1=Transpose[Select[ExtremaAdd[data],
      Chop[(1-#[[2]])]==0&]][[1]];
   a0=Transpose[Select[ExtremaAdd[data],
      Chop[#[[2]]]==0&]][[1]];
   If[Length[a1]>1 || Length[a0]>1,
     Do[Do[If[LessQ[grid[[i]],a0[[j]]],
           If[FreeQ[Transpose[obt][[1]],grid[[i]]],
              obt=Prepend[obt,{grid[[i]],0}]]
        ,{j,Length[a0]}];
        Do[If[GreaterQ[grid[[i]],a1[[j]]],
           If[FreeQ[Transpose[obt][[1]],grid[[i]]],
              obt=Prepend[obt,{grid[[i]],1}]]
        ,{j,Length[a1]}];
     ,{i,Length[grid]}]];
   Do[If[FreeQ[Transpose[obt][[1]],grid[[i]]],
         low=0; high=1;
         tempc=MultiQuadricC[obt,rsq,prec];
         temp=MultiQuadric[grid[[i]],obt,tempc,rsq,prec];
         Do[If[LessQ[obt[[j,1]],grid[[i]]],
               low=Max[low,obt[[j,2]]]];
            If[GreaterQ[obt[[j,1]],grid[[i]]],
               high=Min[high,obt[[j,2]]]];
         ,{j,Length[obt]}];
         If[temp<low,temp=low]; If[temp>high,temp=high];
            obt=Prepend[obt,{grid[[i]],temp}];
            ans[[i]]=temp,ans[[i]]=Select[obt,
              #[[1]]==grid[[i]]&][[1,2]]]
   ,{i,Length[grid]}];
   Transpose[{grid,ans}]]


MLinInt[x_,data_] := Module[{n,xdata,ans,ans2,ans3},
   n=Length[x];
   xdata=Union /@ Transpose[Transpose[data][[1]]];
   ans=ans3=Table[1,{n}];
   ans2=Apply[List,Flatten[Array[Unique[],
      Table[2,{n}],0]],2];
```

```
   Do[While[Not[xdata[[i,ans[[i]]]]<=x[[i]]<=
                        xdata[[i,ans[[i]]+1]]],
      ++ans[[i]]],{i,n}];
   Do[ans3[[i]]=(x[[i]]-xdata[[i,ans[[i]]]])/
      (xdata[[i,ans[[i]]+1]]-xdata[[i,ans[[i]]]]),{i,n}];
   Sum[Select[data,#[[1]]==MapThread[Part,
       {xdata,ans+ans2[[j]]}]&][[1,2]]*
       Product[1-ans2[[j,i]]+(-1)^(1-ans2[[j,i]])*
                         ans3[[i]],{i,n}],{j,2^n}]]


GriddedRMSError[data_,phi_] :=
   Sqrt[Sum[(phi[data[[i,1]]]-data[[i,2]])^2
   ,{i,Length[data]}]/Length[data]]


ShepardRMSError[data_,phi_,m_:10] := Module[{real2,n},
   n=Length[data[[1,1]]];
   real2=PhiGrid[phi,m,n];
   Sqrt[Sum[(real2[[i,2]]-Shepard[real2[[i,1]],data])^2
   ,{i,Length[real2]}]/Length[real2]]]


MultiQuadricRMSError[data_,phi_,m_:10,rsq_:(1/6),
   prec_:$MachinePrecision] :=
Module[{real,c,n},
   n=Length[data[[1,1]]];
   real=PhiGrid[phi,m,n];
   c=MultiQuadricC[data,rsq,prec];
   Sqrt[Sum[(real[[i,2]]-MultiQuadric[real[[i,1]],
      data,c,rsq,prec])^2
   ,{i,Length[real]}]/Length[real]]]


PhiGrid[phi_,m_:4,n_:2] :=
  {#, phi[#]}& /@ GridGenerate[m,n]


RandomPhi[phi_,m_:4,n_:2] :=
  {#, phi[#]}& /@ RandomGenerate[m,n]


RandomGenerate[m_:4,n_:2] := Table[Random[],{m},{n}]
```

```
ShepardGrid[data_,m_:4] := Module[{grid,n,obt},
   n=Length[data[[1,1]]];
   obt=ExtremaAdd[data];
   grid=GridGenerate[m,n];
   Do[grid[[i]]={grid[[i]],Shepard[grid[[i]],obt]}
   ,{i,Length[grid]}];
   grid]


GridGenerate[m_:4,n_:2] :=
  Apply[List,Flatten[Array[Unique[],
     Table[m,{n}],0]],2]/(m-1)
```

J.2 DeterministicAnalysis

```
LessOrEqualQ[a_List,b_List] := FreeQ[Inner[LessEqual,a,b,List],False]


LessQ[a_List,b_List] := FreeQ[Inner[LessEqual,a,b,List],
   False] && MemberQ[Inner[Less,a,b,List],True]


GreaterOrEqualQ[a_List,b_List] := FreeQ[Inner[GreaterEqual,a,b,List],
   False]


GreaterQ[a_List,b_List] := FreeQ[Inner[GreaterEqual,a,b,List],
   False] && MemberQ[Inner[Greater,a,b,List],True]


VectorSpace[p_List] := Module[{tab,size,m,n,a,i,j},
  n=Length[p];
  m=Table[Length[p[[i]]],{i,n}];
  size=Product[m[[i]],{i,n}];
  tab=Table[0,{size}];
  a=Table[1,{i,n}];
  Do[tab[[j]]=Table[p[[i,a[[i]]]],{i,n}];
     Do[If[Take[a,-(n-i)]==Take[m,-(n-i)],
           If[a[[i]]==m[[i]],a[[i]]=1,a[[i]]++]],{i,n-1}];
     If[a[[n]]==m[[n]],a[[n]]=1,a[[n]]++];
  ,{j,size}];
  tab]
```

```
NonDecreasingQ[p_List,phi_Symbol] := Module[{size,m,n,a,b,old,i,j,
                                     iter,ans},
  n=Length[p];
  m=Table[Length[p[[i]]],{i,n}];
  size=Product[m[[i]],{i,n}];
  a=Table[1,{i,n}];
  b=Table[0,{i,n}];
  iter=1;
  ans=True;
  While[iter<=size && ans,
     Do[b[[i]]=p[[i,a[[i]]]],{i,n}];
     old=phi[b];
     Do[
       If[a[[i]]<m[[i]],
         If[phi[ReplacePart[b,p[[i,a[[i]]+1]],i]]<old,
                          ans=False]]
     ,{i,n}];
     Do[If[Take[a,-(n-i)]==Take[m,-(n-i)],
           If[a[[i]]==m[[i]],a[[i]]=1,a[[i]]++]],{i,n-1}];
     If[a[[n]]==m[[n]],a[[n]]=1,a[[n]]++];
     iter++];
  ans]


ProperLimitsQ[p_List,phi_Symbol,pphi_List] :=
   If[phi[Map[Last,p]]==Last[pphi] &&
      phi[Map[First,p]]==First[pphi],True,False]


ReleventComponentsQ[p_List,phi_Symbol] := Module[{vrel,n,size,a,b,m,i,
                                     iter,ans},
  n=Length[p];
  m=Table[Length[p[[i]]],{i,n}];
  size=Product[m[[i]],{i,n}];
  vrel=Table[False,{i,n}];
  a=Table[1,{i,n}];
  b=Table[0,{i,n}];
  iter=1;
  While[iter<=size && Not[FreeQ[vrel,False]],
    Do[b[[i]]=p[[i,a[[i]]]],{i,n}];
    Do[If[vrel[[i]]==False && phi[ReplacePart[b,
                                  p[[i,m[[i]]]],i]]>
          phi[ReplacePart[b,p[[i,1]],i]],vrel[[i]]=True],
       {i,n}];
    Do[If[Take[a,-(n-i)]==Take[m,-(n-i)],
```

```
          If[a[[i]]==m[[i]],a[[i]]=1,a[[i]]++]],{i,n-1}];
    If[a[[n]]==m[[n]],a[[n]]=1,a[[n]]++];
    iter++];
  If[FreeQ[vrel,False],ans=True,ans=False];
  ans]


CoherentQ[p_List,phi_Symbol,pphi_List:{0,1}] :=
   FreeQ[{NonDecreasingQ[p,phi],ProperLimitsQ[p,phi,pphi],
          ReleventComponentsQ[p,phi]},
      False]


LBPFromStructure[p_List,phi_Symbol,fphi_List] :=
Module[{n,m,size,i,a,b,mm,c1,c2,old,iter,ans,locans},
  n=Length[p];
  m=Table[Length[p[[i]]],{i,n}];
  size=Product[m[[i]],{i,n}];
  a=Table[1,{i,n}];
  b=Table[0,{i,n}];
  locans=Table[0,{i,n}];
  iter=1;
  mm=First[fphi];
  ans={};
  While[iter<=size,
     Do[b[[i]]=p[[i,a[[i]]]],{i,n}];
     old=phi[b];
     Do[If[a[[i]]>1,
           locans[[i]]=phi[ReplacePart[b,
              p[[i,a[[i]]-1]],i]],
           locans[[i]]=mm]
     ,{i,n}];
     c2=Position[fphi,old][[1,1]];
     c1=Position[fphi,Max[locans]][[1,1]];
     Do[ans=Append[ans,{b,fphi[[i]],"Lower",
           If[i==c2,"Real","Virtual"]}],{i,c1+1,c2}];
     Do[If[Take[a,-(n-i)]==Take[m,-(n-i)],
           If[a[[i]]==m[[i]],a[[i]]=1,a[[i]]++]],{i,n-1}];
     If[a[[n]]==m[[n]],a[[n]]=1,a[[n]]++];
     iter++];
  Sort[ans,OrderedQ[{#1[[2]], #2[[2]]}]&]]


UBPFromStructure[p_List,phi_Symbol,fphi_List] :=
Module[{n,m,nm,size,i,a,b,c1,c2,old,iter,ans,locans},
```

```
  n=Length[p];
  m=Table[Length[p[[i]]],{i,n}];
  size=Product[m[[i]],{i,n}];
  a=Table[1,{i,n}];
  b=Table[0,{i,n}];
  locans=Table[0,{i,n}];
  iter=1;
  nm=Last[fphi];
  ans={};
  While[iter<=size,
     Do[b[[i]]=p[[i,a[[i]]]],{i,n}];
     old=phi[b];
     Do[If[a[[i]]<m[[i]],
           locans[[i]]=phi[ReplacePart[b,
              p[[i,a[[i]]+1]],i]],
           locans[[i]]=nm]
     ,{i,n}];
     c2=Position[fphi,old][[1,1]];
     c1=Position[fphi,Min[locans]][[1,1]];
     Do[ans=Append[ans,{b,fphi[[i]],"Upper",
           If[i==c2,"Real","Virtual"]}],{i,c2,c1-1}];
     Do[If[Take[a,-(n-i)]==Take[m,-(n-i)],
           If[a[[i]]==m[[i]],a[[i]]=1,a[[i]]++]],{i,n-1}];
     If[a[[n]]==m[[n]],a[[n]]=1,a[[n]]++];
     iter++];
  Sort[ans,OrderedQ[{#1[[2]], #2[[2]]}]&]]


CUVUpperBound[n_Integer,m_Integer] := Coefficient[Sum[u^x,{x,0,m}]^n,
                          u,Floor[(m n + 1)/2]]


BoedigheimerSeriesQ[lbps_List,ubps_List,fphi_List] :=
Module[{good=True,j=1,
        n=Length[lbps[[1,1]]],ubplist,lbplist},
  While[j<=Length[fphi]-1 && good,
    ubplist=Transpose[Select[ubps,
        #[[2]]==fphi[[j]] &]][[1]];
    lbplist=Transpose[Select[lbps,
        #[[2]]==fphi[[j+1]] &]][[1]];
    good=Length[lbplist]==1 && Length[ubplist]==n;
  j++];
  good]
```

```
BoedigheimerParallelQ[lbps_List,ubps_List,fphi_List] :=
Module[{good=True,j=1,
        n=Length[lbps[[1,1]]],ubplist,lbplist},
  While[j<=Length[fphi]-1 && good,
    ubplist=Transpose[Select[ubps,
       #[[2]]==fphi[[j]] &]][[1]];
    lbplist=Transpose[Select[lbps,
       #[[2]]==fphi[[j+1]] &]][[1]];
    good=Length[lbplist]==n && Length[ubplist]==1;
  j++];
  good]



LBPFromPaths[paths_List,n_Integer] :=
  Map[{ReplacePart[Table[0,{n}], 1,
      Partition[#,1]], 1, "Lower", "Real"} &, paths]



UBPFromCuts[cuts_List,n_Integer] :=
  Map[{ReplacePart[Table[1,{n}], 0,
      Partition[#,1]], 0, "Upper", "Real"} &, cuts]



PathsFromLBP[lbps_List] := Map[Flatten[#] &,
  Map[Position[#,1] &, Transpose[lbps][[1]]]]



CutsFromUBP[ubps_List] := Map[Flatten[#] &,
  Map[Position[#,0] &, Transpose[ubps][[1]]]]



BoedigheimerReleventComponentsQ[lbps_List,ubps_List,p_List] :=
Module[{good=True,j=1,n=Length[p],l,u},
  l=Transpose[Transpose[lbps][[1]]];
  u=Transpose[Transpose[ubps][[1]]];
  While[j<=n && good,
    good = MemberQ[First[p[[j]]]<# & /@ l[[j]],True]  ||
           MemberQ[Last[p[[j]]]># & /@ u[[j]],True];
  j++];
  good]
```

```
StructuralImportances[p_List,phi_Symbol] := Module[{n,temp,cnt,ans,size},
  n=Length[p];
  ans=Table[1,{n}];
  size=Times @@ (Length[#]& /@ p);
  Do[temp=VectorSpace[ReplacePart[p,{Last[p[[i]]]},i]];
     cnt=0;
     Do[If[phi[temp[[j]]]>phi[
           ReplacePart[temp[[j]],First[p[[i]]],i]],cnt++]
        ,{j,Length[temp]}];
     ans[[i]]=cnt*Length[p[[i]]]/size;
     ,{i,n}];
  ans]



UBPToLBP[ubps_List,p_List,fphi_List] :=
Module[{potsols,ubplist,temp,ans={}},
  Do[ubplist=Select[ubps,#[[2]]==fphi[[k]] &];
     potsols=utlposs[ubplist,p];
     potsols=utludomm[potsols,ubplist];
     potsols=utlldomm[potsols];
     temp=potsols;
     Do[temp[[i]]={potsols[[i]],fphi[[k+1]],
         "Lower","Indet"},
     {i,Length[potsols]}];
     ans=Append[ans,temp];
  ,{k,Length[fphi]-1}];
  ans=Flatten[ans,1];
  Sort[ans,OrderedQ[{#1[[2]], #2[[2]]}]&]]



utlposs[ubplist_List,p_List] :=
Module[{a=p,toperm=p,ans={},n=Length[p]},
     Do[Do[a[[i]]=Position[p[[i]],
           ubplist[[j,1,i]]][[1,1]],{i,n}];
        Do[If[a[[i]]<Length[p[[i]]],
              Do[toperm[[i2]]=Take[p[[i2]],
                 a[[i2]]],{i2,n}];
              toperm[[i]]={p[[i,a[[i]]+1]]};
              ans=Append[ans,VectorSpace[toperm]]]
        ,{i,n}]
     ,{j,Length[ubplist]}];
     Union[Flatten[ans,1]]]
```

```
utludomm[potsols_List,ubplist_List] :=
Module[{go,j,a={}},
     Do[go=False; j=1;
        While[go==False && j<=Length[ubplist],
           If[LessOrEqualQ[potsols[[l]],ubplist[[j,1]]],go=True];
           j++];
        If[go==True,a=Append[a,l]];
     ,{l,1,Length[potsols]}];
     Complement[potsols,potsols[[a]]]]


utlldomm[potsols_List] :=
Module[{go,j,ans=potsols,n=Length[potsols]},
        Do[go=False; j=1;
           While[go==False && j<=Length[ans],
              If[GreaterQ[ans[[l]],ans[[j]]],go=True];
              j++];
           If[go==True,ans=Drop[ans,{l}]];
        ,{l,n,1,-1}];
     ans]


LBPToUBP[lbps_List,p_List,fphi_List] :=
Module[{potsols,lbplist,temp,ans={}},
  Do[lbplist=Select[lbps,#[[2]]==fphi[[k]] &];
     potsols=ltlposs[lbplist,p];
     potsols=ltludomm[potsols,lbplist];
     potsols=ltlldomm[potsols];
     temp=potsols;
     Do[temp[[i]]={potsols[[i]],
       fphi[[k-1]],"Upper","Indet"},
     {i,Length[potsols]}];
     ans=Append[ans,temp];
  ,{k,2,Length[fphi]}];
  ans=Flatten[ans,1];
  Sort[ans,OrderedQ[{#1[[2]], #2[[2]]}]&]]


ltlposs[lbplist_List,p_List] :=
Module[{a=p,toperm=p,ans={},n=Length[p]},
     Do[Do[a[[i]]=Position[p[[i]],
           lbplist[[j,1,i]]][[1,1]],{i,n}];
        Do[If[a[[i]]>1,
              Do[toperm[[i2]]=Take[p[[i2]],
                 a[[i2]]-Length[p[[i2]]]-1],
```

```
                {i2,n}];
              toperm[[i]]={p[[i,a[[i]]-1]]};
              ans=Append[ans,VectorSpace[toperm]]]
        ,{i,n}]
     ,{j,Length[lbplist]}];
     Union[Flatten[ans,1]]]


ltludomm[potsols_List,lbplist_List] :=
Module[{go,j,a={}},
     Do[go=False; j=1;
        While[go==False && j<=Length[lbplist],
           If[GreaterOrEqualQ[potsols[[l]],lbplist[[j,1]]],go=True];
           j++];
        If[go==True,a=Append[a,l]];
     ,{l,1,Length[potsols]}];
     Complement[potsols,potsols[[a]]]]


ltlldomm[potsols_List] :=
Module[{go,j,ans=potsols,n=Length[potsols]},
        Do[go=False; j=1;
           While[go==False && j<=Length[ans],
              If[LessQ[ans[[l]],ans[[j]]],go=True];
              j++];
           If[go==True,ans=Drop[ans,{l}]];
        ,{l,n,1,-1}];
     ans]


CutsToPaths[cuts_List, n_Integer] := PathsFromLBP[UBPToLBP[
  UBPFromCuts[cuts,n],Table[{0,1},{n}],{0,1}]]


PathsToCuts[paths_List, n_Integer] := CutsFromUBP[LBPToUBP[
  LBPFromPaths[paths,n],Table[{0,1},{n}],{0,1}]]


SystemStateFromLBP[lbps_List,fphi_List,x_List] :=
Module[{good=True,j=2,lbplist,ans},
  While[j<=Length[fphi] && good,
    lbplist=Transpose[Select[lbps,
       #[[2]]==fphi[[j]] &]][[1]];
    good=MemberQ[(x~GreaterOrEqualQ~#)& /@ lbplist, True];
  j++];
  If[good,ans=Last[fphi],ans=fphi[[j-2]]];
  ans]
```

```
SystemStateFromUBP[ubps_List,fphi_List,x_List] :=
  Module[{good=True,j=Length[fphi]-1,ubplist,ans},
  While[j>=1 && good,
    ubplist=Transpose[Select[ubps,
       #[[2]]==fphi[[j]] &]][[1]];
    good=MemberQ[(x~LessOrEqualQ~#)& /@ ubplist, True];
  j--];
  If[good,ans=First[fphi],ans=fphi[[j+2]]];
  ans]


BPClean[bplist_List,strone_String,strtwo_String:"Indet"] :=
Module[{ans=bplist,temp},
  ans=Sort[bplist,OrderedQ[{#1[[2]], #2[[2]]}]&];
  Do[If[Length[ans[[i]]]==2,
            ans[[i]]=Append[ans[[i]],strone];

            ans[[i]]=Append[ans[[i]],strtwo]];
     If[Length[ans[[i]]]==3,temp=ans[[i,3]];
        If[temp=="Upper" || temp=="Lower",
           ans[[i]]=Append[ans[[i]],strtwo]];
        If[temp==0,
           ans[[i]]=Append[ans[[i]],"Real"];
           ans[[i,3]]=strone];
        If[temp==1,
           ans[[i]]=Append[ans[[i]],"Virtual"];
           ans[[i,3]]=strone];
        If[temp==2,
           ans[[i]]=Append[ans[[i]],"Indet"];
           ans[[i,3]]=strone];
        If[temp=="Real" || temp=="Virtual",
           ans[[i,3]]=strone;
           ans[[i]]=Append[ans[[i]],temp]]]
  ,{i,Length[bplist]}];
ans]


LBPSelfConsistentQ[lbpsin_List] := Module[{i=1,good=True,
   lbps=Transpose[lbpsin][[1]]}, While[i<Length[lbps] &&
   (good=FreeQ[Map[GreaterQ[lbps[[i]],#] &, Drop[lbps,i]],True]),
   i++]; good]


UBPSelfConsistentQ[ubpsin_List] := Module[{i=Length[ubps],good=True,
   ubps=Transpose[ubpsin][[1]]},
```

```
While[i>1 && (good=FreeQ[Map[LessQ[ubps[[i]],#] &, Take[ubps,
   i-1]],True]), i--];
good]



BPConsistentToEachOtherQ[lbps_List,ubps_List] := Module[{i=1,good=True},
While[i<=Length[ubps] && (good=FreeQ[Map[GreaterOrEqualQ[ubps[[i,1]],
#[[1]]] &, Select[lbps,#[[2]]>ubps[[i,2]] &]],True]), i++];
good]



SystemLimitsFromBP[lbps_List,ubps_List] :=
  {Min[Transpose[ubps][[2]]],
   Max[Transpose[lbps][[2]]]}



StructureFromPhi[p_List,phi_Symbol] := {#, phi[#]}& /@ VectorSpace[p]



StructureFromLBP[p_List,lbps_List,fphi_List] :=
  {#, SystemStateFromLBP[lbps,fphi,#]}& /@ VectorSpace[p]



StructureFromUBP[p_List,ubps_List,fphi_List] :=
  {#, SystemStateFromUBP[ubps,fphi,#]}& /@ VectorSpace[p]



SystemSpaceFromBP[lbps_List,ubps_List] :=
    Union[Transpose[ubps][[2]], Transpose[lbps][[2]]]



BPTypeFind[bplist_List, phi_Symbol] :=
   Module[{ans=bplist},
      Do[If[phi[ans[[i,1]]]==ans[[i,2]],
            ans[[i,4]]="Real",ans[[i,4]]="Virtual"]
      ,{i,Length[bplist]}];
   ans]
```

J.3 Distributions

The functions in this section require that the Statistics‘ContinuousDistributions and Statistics‘DiscreteDistributions packages also be loaded.

```
rdc[n_, t_] := 1-CDF[ChiDistribution[n],t]


rdcs[n_, t_] := 1-CDF[ChiSquareDistribution[n],t]


rde[lambda_, t_] := 1-CDF[ExponentialDistribution[lambda],t]


rdfr[n1_, n2_, t_] := 1-CDF[FRatioDistribution[n1,n2],t]


rdg[alpha_, beta_, t_] := 1-CDF[GammaDistribution[alpha, beta],t]


rdhn[theta_, t_] := 1-CDF[HalfNormalDistribution[theta], t]


rdln[mu_, sigma_, t_] := 1-CDF[LogNormalDistribution[mu,
   sigma], t]


rdncs[n_, lambda_, t_] :=
   1-CDF[NoncentralChiSquareDistribution[n,lambda],t]


rdnfr[n1_, n2_, lambda_, t_] :=
   1-CDF[NoncentralFRatioDistribution[n1,n2,lambda],t]


rdr[sigma_, t_] := 1-CDF[RayleighDistribution[sigma], t]


rdw[alpha_, beta_, t_]  :=
   1-CDF[WeibullDistribution[alpha, beta], t]


rdu[max_, t_]  := 1-CDF[UniformDistribution[0, max], t]


pdc[n_, t_] := PDF[ChiDistribution[n],t]
```

```
pdcs[n_, t_] := PDF[ChiSquareDistribution[n],t]


pde[lambda_, t_] := PDF[ExponentialDistribution[lambda],t]


pdfr[n1_, n2_, t_] := PDF[FRatioDistribution[n1,n2],t]


pdg[alpha_, beta_, t_] := PDF[GammaDistribution[alpha, beta], t]


pdhn[theta_, t_] := PDF[HalfNormalDistribution[theta], t]


pdln[mu_, sigma_, t_] := PDF[LogNormalDistribution[mu, sigma], t]


pdncs[n_, lambda_, t_] :=
   PDF[NoncentralChiSquareDistribution[n,lambda],t]


pdnfr[n1_, n2_, lambda_, t_] :=
   PDF[NoncentralFRatioDistribution[n1,n2,lambda],t]


pdr[sigma_, t_] := PDF[RayleighDistribution[sigma], t]


pdw[alpha_, beta_, t_]  :=
   PDF[WeibullDistribution[alpha, beta], t]


pdu[max_, t_]  := PDF[UniformDistribution[0, max], t]


BinaryCDF[x_, p_] := Which[
          x<0,  0,
          x>=1, 1,
          True, 1-p
         ]
```

```
MultistateCDF[x_, sys_] := Which[
              x<First[sys][[1]], 0,
              x>=Last[sys][[1]], 1,
              x>=First[sys][[1]] && x<Last[sys][[1]],
                      Do[If[sys[[i,1]]<=x<sys[[i+1,1]],
                         Return[Sum[sys[[j,2]],{j,i}]]]
                      ,{i,Length[sys]-1}]]


CountableInfinityCDF[x_, mmadist_] :=
    Which[x<0,  0,
          x>=1, 1,
          True, CDF[mmadist,1/(1-x)-1]]


UniformCDF[x_] := UniformMixedCDF[x, 0, 0]


TriangularCDF[x_, a_:1/2] := Which[x<0, 0,
                            x>=1, 1,
                            x>=0 && x<a, x^2 / a,
                            True, (x^2 -2x+a)/(a-1)]


NonTruncatedCDF[x_, mmadist_] :=
   (CDF[mmadist,x]-CDF[mmadist,0])/
   (CDF[mmadist,1]-CDF[mmadist,0])


UniformMixedCDF[x_, li_:0, ui_:0] := Which[
           x<0,  0,
           x>=1, 1,
           True, li+x(1-li-ui)
          ]


TruncatedCDF[x_, mmadist_] := Which[
             x<0,  0,
             x>=1, 1,
             True, CDF[mmadist,x]
            ]


UniformPDF[x_,min_:0,max_:1] := Which[x<min, 0,
                           x>max, 0,
                           True, 1/(max-min)]
```

```
TriangularPDF[x_, a_:(1/2), min_:0, max_:1] :=
   Which[x<min,0,
         x>max,0,
         x>=min && x<=a, 2/(max-min) (x-min)/(a-min),
         x>=a && x<=max, 2/(max-min) (1 - (x-a)/(max-a))]

BernoulliSYS[p_] := {{0, 1-p}, {1, p}}

UniformDiscreteSYS[n_] :=
   If[n==1,{{0,1}},Table[{(i-1)/(n-1),1/n},{i,n}]]

BinomialSYS[n_, p_] := Table[{i/n,
    PDF[BinomialDistribution[n, p], i]}, {i, 0, n}]

HypergeometricSYS[n_, nsucc_, ntot_] :=
   With[{max=Min[n, nsucc], min=Max[0, n+nsucc-ntot]},
     Table[{(i-min)/(max-min),
            PDF[HypergeometricDistribution[n, nsucc, ntot], i]}
     ,{i, min, max}]
]

FnEstimate[list_] := Module[{ans, n=Length[list]},
   ans={#, #}& /@ Union[list];
   Do[ans[[i,2]]=Length[Select[list, # == ans[[i,1]]&]]/n,
        {i,Length[ans]}];
   ans]

FnEstimateVar[list_] := 1/(4 Length[list])

CustomerLimitsGamma[m_, x_:(1/2), a_:0, b_:0] :=
   Module[{v,ans,b2},
   If[b==0,b2=m,b2=b];
   ans = v /. FindRoot[CDF[GammaDistribution[v,
          m/(v-1)],b2]-
          CDF[GammaDistribution[v,
          m/(v-1)],a] == x,{v,1.1,1.2}];
    {ans, m/(ans-1)}]

CustomerLimitsWeibull[m_, x_:(1/2), a_:0, b_:0] :=
   Module[{v,ans,b2},
   If[b==0,b2=m,b2=b];
   ans = v /. FindRoot[CDF[WeibullDistribution[v,
          m/((v-1)/v)^(1/v)],b2]-
          CDF[WeibullDistribution[v,
          m/((v-1)/v)^(1/v)],a] == x,{v,1.1,1.2}];
   {ans,m/((ans-1)/ans)^(1/ans)}]
```

J.4 DynamicModels

The functions in this section require that the Calculus‘LaplaceTransform package also be loaded.

```
PDPErlangian[M_Integer, val_, t_] := Module[{ans=Table[0,{M+1}]},
   Do[ans[[i+1]]=
        (val t)^(M - i)*E^(-val t)/(M-i)!,{i,1,M}];
   ans[[1]]=1-Sum[ans[[i+1]],{i,1,M}];
   ans]



PDPAdjacent[mymu_List, t_] := Module[{m, mu},
    m=Length[mymu]; mu=Prepend[mymu,0];
    Table[pti[i,m,mu,t], {i,0,m}]]



PDPNonAdjacent[n_Integer, mu_, t_] := Module[{j, hz2, a={}},
   hz2=Array[mu,{20,20},0];
   For[j=n, j>=0, j--, a=Prepend[a,pdp2[hz2,j,n,t]]]; a]



pti[i_Integer, m_Integer, mu_List, t_] := Sum[aij[i,j,m,mu]*
    E^(-mu[[(j)+1]]*t), {j,i,m}]*Product[mu[[(j)+1]],{j,i+1,m}]



aij[i_Integer, j_Integer, m_Integer, mu_List] := Product[
   If[k=!=j,1/(mu[[(k)+1]]-mu[[(j)+1]]),1],{k, i, m}]



pdp2[hz2_, n_, m_, t_] :=InverseLaplaceTransform[pdpz2[hz2,n,m],z,t]



pdpz2[hz2_, m_, m_] := 1/(Sum[hz2[[m+1,hz2temp]],
   {hz2temp, 1, m}] + z)
pdpz2[hz2_, n_, m_] := Sum[hz2[[i + 1,n + 1]]*pdpz2[hz2, i, m],
   {i, n + 1, m}]/(Sum[hz2[[n+1,hz2temp2]],
   {hz2temp2,1,n}] + z)
```

```
CTMCStateProbabilities[matrix_List, t_, initialstate_Integer:-1] :=
   Module[{n = Dimensions[matrix][[1]], m = matrix,
           initial = initialstate, b, v, i, z, k},
      If[initial == -1, initial = n - 1]; If[initial == 0,
         k = n, k = 1];
      Do[m[[i,i]] = 0, {i, n}];
      For[i = 1, i <= n, i++, v = Plus @@ Transpose[m][[i]];
         m[[i,i]] = -v];
    m = m - z*IdentityMatrix[n]; m[[k]] = Table[1, {n}];
    b = Table[0, {n}]; b[[k]] = 1/z; b[[initial + 1]] = -1;
    InverseLaplaceTransform[LinearSolve[m, b], z, t]]


CTMCMeanAbsorptionTimes[matrix_List, absorbingstates_List:{},
         initialstate_Integer:-1] :=
  Module[{n = Dimensions[matrix][[1]], m = matrix,
          initial = initialstate, b, v, i},
    For[i=1,i<=Dimensions[absorbingstates][[1]],i++,
        m=Transpose[ReplacePart[
          Transpose[m],Table[0,{n}],absorbingstates[[i]]+1]]];
    If[initial == -1, initial = n - 1];
    Do[m[[i,i]] = 0, {i, n}];
    For[i = 1, i <= n, i++, v = Plus @@ Transpose[m][[i]];
        m[[i,i]] = -v];
    b = Table[0, {n}]; b[[initial + 1]] = -1;
    For[i=n,i>=1,i--,If[Transpose[m][[i,i]]==0,
          m=Drop[m,{i,i}];
        m=Transpose[Drop[Transpose[m],{i,i}]];
        b=Drop[b,{i,i}]]];
    Plus @@ LinearSolve[m, b]]


CTMCSteadyStateProbabilities[matrix_List] :=
  Module[{n = Dimensions[matrix][[1]], m = matrix, b, v, i},
    Do[m[[i,i]] = 0, {i, n}];
    For[i = 1, i <= n, i++, v = Plus @@ Transpose[m][[i]];
       m[[i,i]] = -v];
    m[[1]] = Table[1, {n}];
    b = Table[0, {n}];
    b[[1]]=1;
    LinearSolve[m, b]]


CTMCMeanArrivals[m_List]:=Module[{n=Dimensions[m][[1]], v, i},
  v = Table[0, {n}];
```

```
  For[i = 1, i <= n, i++,
     v[[i]] = Plus @@ Transpose[m][[i]] - m[[i,i]]];
  v*CTMCSteadyStateProbabilities[m]]


CTMCStayDurations[m_List]:=Module[{n=Dimensions[m][[1]], v, i},
  v = Table[0, {n}];
  For[i = 1, i <= n, i++,
     v[[i]] = Plus @@ Transpose[m][[i]] - m[[i,i]]];
     1/v]
```

J.5 Measures

```
ExpectedFnState[sys_List, f_] := Plus @@ ((f[#[[1]]] #[[2]])& /@ sys)


ExpectedState[sys_List] := Plus @@ ((#[[1]] #[[2]])& /@ sys)


CDFExpectedState[cdf_, x_, max_:1] := CDFMoment[cdf, x, 1, max]


ExpectedOutput[sys_List, t_, tstar_, ni_:False] :=
   If[ni,NIntegrate[ExpectedState[sys], {t, 0, tstar}],
          Integrate[ExpectedState[sys], {t, 0, tstar}]]


CDFExpectedOutput[cdf_, x_, t_, tstar_, max_:1] :=
   NIntegrate[CDFExpectedState[(cdf /. t->t0), x, max], {t0, 0, tstar}]


ExpectedTotalOutput[sys_List, t_, ni_:False] :=
   ExpectedOutput[sys, t, Infinity, ni]


CDFExpectedTotalOutput[cdf_, x_, t_, max_:1] :=
   CDFExpectedOutput[cdf, x, t, Infinity, max]


VarianceOfOutputUB[sys_List, t_, tstar_, ni_:False] :=
   If[ni,NIntegrate[(ExpectedSquaredSystemState[sys])^(1/2),
        {t,0,tstar}]^2-
                    ExpectedOutput[sys,t,tstar,ni]^2,
         Integrate[(ExpectedSquaredSystemState[sys])^(1/2),
                   {t,0,tstar}]^2-
                    ExpectedOutput[sys,t,tstar,ni]^2]
```

```
CDFVarianceOfOutputUB[cdf_, x_, t_, tstar_, max_:1] :=
   NIntegrate[(CDFMoment[(cdf /. t->t0), x, 2, max])^(1/2),
        {t0,0,tstar}]^2  -
   CDFExpectedOutput[cdf,x,t,tstar,max]^2


UpperStatesProbability[sys_List, j_] := Plus @@ ((#[[2]]&) /@
    Select[sys, #[[1]]>=j &])


CDFUpperStatesProbability[cdf_, x_, j_] :=
   CDFR[cdf, x, j]+CDFStateProbability[cdf, x, j]


StateDwellTime[sys_List, t_, j_, ni_:False] :=
  If[ni,NIntegrate[StateProbability[sys,j], {t, 0, Infinity}],
        Integrate[StateProbability[sys,j], {t, 0, Infinity}]]


CDFStateDwellTime[cdf_, x_, j_, t_] :=
   NIntegrate[(CDFStateProbability[cdf, x, j] /. t->t0) //
      Evaluate, {t0, 0, Infinity}]


StateVariance[sys_List] :=
   ExpectedSquaredSystemState[sys]-(ExpectedState[sys])^2


CDFStateVariance[cdf_, x_, max_:1] :=
    CDFMoment[cdf, x, 2, max]-(CDFMoment[cdf,x,1,max])^2


LifetimeWeighted[sys_List, t_, u_, utotal_:1, ni_:False] :=
   If[ni,NIntegrate[u*ExpectedState[sys],{t,0,Infinity}]/utotal,
          Integrate[u*ExpectedState[sys],{t,0,Infinity}]/utotal]


CDFLifetimeWeighted[cdf_, x_, t_, u_, utotal_:1, max_:1] :=
NIntegrate[(u*CDFExpectedState[(cdf /. t->t0), x, max] /. t->t0),
    {t0, 0, Infinity}]/utotal


StateProbability[sys_List, j_] :=
   Select[sys, #[[1]]==j &, 1][[1,2]]

CDFStateProbability[cdf_, x_, x0_:1] :=
   Chop[CDFF[cdf,x,x0]-CDFF[cdf,x,x0-100*$MachineEpsilon]]
```

```
UpperStatesDwellTime[sys_List, t_, j_, ni_:False] :=
   If[ni,NIntegrate[SYSR[sys,j], {t, 0, Infinity}],
          Integrate[SYSR[sys,j],
                       {t, 0, Infinity}]]


CDFUpperStatesDwellTime[cdf_, x_, j_, t_] :=
   NIntegrate[(CDFR[cdf, x, j] /. t->t0) // Evaluate,
        {t0, 0, Infinity}]


LowerStatesProbability[sys_List, j_] :=
   1-UpperStatesProbability[sys,j]


CDFLowerStatesProbability[cdf_, x_, j_] :=
   1-CDFUpperStatesProbability[cdf, x, j]


RangeStatesProbability[sys_List, j_, k_] :=
   Plus @@ ((#[[2]]&) /@
      Select[sys, #[[1]]>j && #[[1]]<=k &])


CDFRangeStatesProbability[cdf_, x_, j_, k_] :=
   CDFF[cdf, x, k] - CDFF[cdf, x, j]


ExpectedLostOutput[sys_List, t_, tstar_, ni_:False] :=
   sys[[Length[sys],1]]*tstar-ExpectedOutput[sys,t,tstar,ni]


CDFExpectedLostOutput[cdf_, x_, t_, tstar_, max_:1] :=
   max*tstar-CDFExpectedOutput[cdf, x, t, tstar, max]


ExpectedScaledOutput[sys_List, t_, tstar_, ni_:False] :=
   (ExpectedOutput[sys,t,tstar,ni] - tstar*sys[[1,1]])/
   (tstar*sys[[Length[sys],1]] - tstar*sys[[1,1]])


CDFExpectedScaledOutput[cdf_, x_, t_, tstar_, max_:1] :=
   CDFExpectedOutput[cdf, x, t, tstar, max]/(max*tstar)


StateStandardDeviation[sys_List] := (StateVariance[sys])^(1/2)
```

```
CDFStateStandardDeviation[cdf_, x_, max_:1] :=
   (CDFStateVariance[cdf, x, max])^(1/2)


ExpectedSquaredSystemState[sys_List] :=
   Plus @@ ((#[[1]]^2 #[[2]])& /@ sys)


DerivativeOfLSP[sys_List, t_, j_] :=
   D[LowerStatesProbability[sys, j], t]


CDFDerivativeOfLSP[cdf_, x_, j_, t_] :=
   D[CDFLowerStatesProbability[cdf, x, j], t]


DegradationRate[sys_List, t_, j_] :=
   DerivativeOfLSP[sys, t,j]/UpperStatesProbability[sys, j]


Hazard[sys_List, t_, j_] :=
   D[SYSF[sys,j], t]/SYSR[sys, j]


CDFHazard[cdf_, x_, x0r_, t_, t0r_, prec_:$MachinePrecision] :=
   With[{h=1/10^(prec-6),
         t0=SetPrecision[t0r,prec],
         f=CDFF[cdf, x, SetPrecision[x0r,prec]],
         r=CDFR[cdf, x, SetPrecision[x0r,prec]]},
   N[(((f /. t->(t0+h)) - (f /. t->t0))/(h (r /. t->t0))), prec]]


CDFHazardB[cdf_, x_, x0r_, t_, t0r_, prec_:$MachinePrecision] :=
   Module[{t0,F,R},
   Needs["NumericalMath`NLimit`"];
            t0=SetPrecision[t0r,prec];
           F=CDFF[cdf, x, SetPrecision[x0r,prec]];
           R=CDFR[cdf, x, SetPrecision[x0r,prec]] /. t->t0;
   ND[F,t,t0, WorkingPrecision->prec]/R]


DerivativeOfExpectedState[sys_List, t_] :=
   D[ExpectedState[sys],t]
```

```
CDFDerivativeOfExpectedState[cdf_, x_, t_, t0r_,
       prec_:$MachinePrecision, max_:1] :=
   Module[{h=1/(2 10^(prec-6)),
           t0=SetPrecision[t0r,prec], fh, f},
          fh=CDFExpectedState[(cdf /. t->(t0+h)), x, max];
          f=CDFExpectedState[(cdf /. t->(t0-h)), x, max];
   N[((fh-f)/(2 h)), prec]]


CumulativeStandardDeviation[sys_List, t_, tstar_, ni_:False] :=
   If[ni,NIntegrate[StateStandardDeviation[sys], {t, 0, tstar}],
          Integrate[StateStandardDeviation[sys], {t, 0, tstar}]]


CDFCumulativeStandardDeviation[cdf_, x_, t_, tstar_, max_:1] :=
   NIntegrate[CDFStateStandardDeviation[(cdf /. t->t0), x, max],
        {t0, 0, tstar}]


SYSF[sys_List, j_] := Plus @@ ((#[[2]]&) /@
    Select[sys, #[[1]]<=j &])


CDFF[cdf_, x_, x0_:0] := cdf /. x->x0


SYSR[sys_List, j_] := Plus @@ ((#[[2]]&) /@
    Select[sys, #[[1]]>j &])


CDFR[cdf_, x_, x0_:1] := 1-CDFF[cdf, x, x0]


ChebyshevUB[x_, var_, epsilon_] := var/epsilon^2


Moment[sys_List, n_:1] := Module[{f}, f[x_] := x^n;
   ExpectedFnState[sys, f]]


CDFMoment[cdf_, x_, n_:1, max_:1] :=
   NIntegrate[n x^(n-1) (1-cdf),{x,0,max}]


MaximalStateProportion[sys_List, t_, tstar_, ni_:False] :=
  If[ni,NIntegrate[sys[[Length[sys],2]]/tstar, {t, 0, tstar}],
         Integrate[sys[[Length[sys],2]]/tstar,
            {t, 0, tstar}]]
```

```
CDFMaximalStateProportion[cdf_, x_, t_, tstar_, max_:1] :=
  NIntegrate[((CDFStateProbability[cdf, x, max]/tstar) /.
              t->t0) // Evaluate, {t0, 0, tstar}]


OnStreamAvailability[sys_List, t_, tstar_, j_, ni_:False] :=
  If[ni,NIntegrate[SYSR[sys,j]/tstar, {t, 0, tstar}],
         Integrate[SYSR[sys,j]/tstar,
            {t, 0, tstar}]]


CDFOnStreamAvailability[cdf_, x_, j_, t_, tstar_] :=
  NIntegrate[((CDFR[cdf, x, j]/tstar) /. t->t0) //
     Evaluate, {t0, 0, tstar}]


DiscreteEntropy[sys_] :=
  - Plus @@ ((# Log[#])& /@ Select[(#[[2]]& /@ sys), Positive])


ContinuousEntropy[f_, x_, min_:0, max_:1, ni_:False] :=
If[ni, -NIntegrate[f*Log[f], {x, min, max}],
        -Integrate[f*Log[f], {x, min, max}]]


SYSSkewness[sys_] :=
  With[{m={Moment[sys,1],
           Moment[sys,2],
           Moment[sys,3]}},
  (m[[3]] - 3 m[[1]] m[[2]] + 2 (m[[1]])^3)/
  (m[[2]] - (m[[1]])^2)^(3/2)]


CDFSkewness[cdf_, x_, max_:1] :=
  With[{m={CDFMoment[cdf,x,1,max],
           CDFMoment[cdf,x,2,max],
           CDFMoment[cdf,x,3,max]}},
  (m[[3]] - 3 m[[1]] m[[2]] + 2 (m[[1]])^3)/
  (m[[2]] - (m[[1]])^2)^(3/2)]


SYSKurtosis[sys_] :=
  With[{m={Moment[sys,1],
           Moment[sys,2],
           Moment[sys,3],
           Moment[sys,4]}},
  (m[[4]] - 4 m[[1]] m[[3]] + 6 (m[[1]])^2 m[[2]] - 3 (m[[1]])^4)/
  (m[[2]] - (m[[1]])^2)^2]
```

```
CDFKurtosis[cdf_, x_, max_:1] :=
  With[{m={CDFMoment[cdf,x,1,max],
           CDFMoment[cdf,x,2,max],
           CDFMoment[cdf,x,3,max],
           CDFMoment[cdf,x,4,max]}},
  (m[[4]] - 4 m[[1]] m[[3]] + 6 (m[[1]])^2 m[[2]] - 3 (m[[1]])^4)/
  (m[[2]] - (m[[1]])^2)^2]


SYSKurtosisExcess[sys_] := SYSKurtosis[sys] - 3


CDFKurtosisExcess[cdf_, x_, max_:1] := CDFKurtosis[cdf, x, max] - 3


SYSQuantile[sys_, q_:1/2] := Module[{temp=0},
     Do[temp += sys[[i,2]];
        If[temp>=q, Return[sys[[i,1]]]],
     {i, Length[sys]}]]


CDFQuantile[cdf_, x_, q_:1/2, prec_:$MachinePrecision,
    min_:0, max_:1] :=
    (CDFQuantileUp[cdf, x, q, prec, min, max]+
     CDFQuantileDown[cdf, x, q, prec, min, max])/2


CDFQuantileDown[cdf_, x_, q_:1/2, prec_:$MachinePrecision,
               min_:0, max_:1] :=
   Module[{tol=10^(-prec+3), u=max, d=min, temp},
   While[u-d>=tol,
      temp=N[(u-d)/2 + d, prec];
      If[(cdf /. x->temp) < q, d=temp, u=temp]];
   Which[Chop[temp-min,tol]==0,min,
         Chop[temp-max,tol]==0,max,
         True, temp]]


CDFQuantileUp[cdf_, x_, q_:1/2, prec_:$MachinePrecision,
               min_:0, max_:1] :=
   Module[{tol=10^(-prec+3), u=max, d=min, temp},
   While[u-d>=tol,
      temp=N[(u-d)/2 + d, prec];
      If[(cdf /. x->temp) <= q, d=temp, u=temp]];
   Which[Chop[temp-min,tol]==0,min,
         Chop[temp-max,tol]==0,max,
         True, temp]]
```

```
SYSMedian[sys_] := SYSQuantile[sys, 1/2]


CDFMedian[cdf_, x_, min_:0, max_:1] :=
    CDFQuantile[cdf, x, 1/2, $MachinePrecision, min, max]


SYSQuartiles[sys_] :=
   {SYSQuantile[sys, 1/4],
    SYSQuantile[sys, 1/2],
    SYSQuantile[sys, 3/4]}


CDFQuartiles[cdf_, x_, min_:0, max_:1] :=
   {CDFQuantile[cdf, x, 1/4, $MachinePrecision, min, max],
    CDFQuantile[cdf, x, 1/2, $MachinePrecision, min, max],
    CDFQuantile[cdf, x, 3/4, $MachinePrecision, min, max]}


SYSQuadraticRaw[sys_, y0_, k_:1] := Module[{ans=sys},
   Do[ans[[i,1]]=k (sys[[i,1]]-y0)^2,{i,Length[sys]}]; ans]


CDFQuadraticMean[cdf_, x_, y0_, k_:1, max_:1] :=
   With[{m=Table[CDFMoment[cdf, x, i, max], {i,2}]},
   k*(y0^2 - 2*y0*m[[1]] + m[[2]])
   ]


CDFQuadraticVariance[cdf_, x_, y0_, k_:1, max_:1] :=
   With[{m=Table[CDFMoment[cdf, x, i, max], {i,4}]},
   k^2*(-4*y0^2*m[[1]]^2 + 4*y0^2*m[[2]] + 4*y0*m[[1]]*m[[2]] -
      m[[2]]^2 - 4*y0*m[[3]] + m[[4]])
   ]


CDFQuadraticSkewness[cdf_, x_, y0_, k_:1, max_:1] :=
   With[{m=Table[CDFMoment[cdf, x, i, max], {i,6}]},
   (2*k^3*(y0^2 - 2*y0*m[[1]] + m[[2]])^3 -
     3*k^3*(y0^2 - 2*y0*m[[1]] + m[[2]])*
      (y0^4 - 4*y0^3*m[[1]] + 6*y0^2*m[[2]] - 4*y0*m[[3]] + m[[4]]) +
     k^3*(y0^6 - 6*y0^5*m[[1]] + 15*y0^4*m[[2]] - 20*y0^3*m[[3]] +
        15*y0^2*m[[4]] - 6*y0*m[[5]] + m[[6]]))/
  (k^2*(-4*y0^2*m[[1]]^2 + 4*y0^2*m[[2]] + 4*y0*m[[1]]*m[[2]] - m[[2]]^2 -
        4*y0*m[[3]] + m[[4]]))^(3/2)
   ]
```

```
CDFQuadraticKurtosis[cdf_, x_, y0_, k_:1, max_:1] :=
   With[{m=Table[CDFMoment[cdf, x, i, max], {i,8}]},
   (-3*k^4*(y0^2 - 2*y0*m[[1]] + m[[2]])^4 +
     6*k^4*(y0^2 - 2*y0*m[[1]] + m[[2]])^2*
      (y0^4 - 4*y0^3*m[[1]] + 6*y0^2*m[[2]] - 4*y0*m[[3]] + m[[4]]) -
     4*k^4*(y0^2 - 2*y0*m[[1]] + m[[2]])*
      (y0^6 - 6*y0^5*m[[1]] + 15*y0^4*m[[2]] - 20*y0^3*m[[3]] +
        15*y0^2*m[[4]] - 6*y0*m[[5]] + m[[6]]) +
     k^4*(y0^8 - 8*y0^7*m[[1]] + 28*y0^6*m[[2]] - 56*y0^5*m[[3]] +
  70*y0^4*m[[4]] - 56*y0^3*m[[5]] + 28*y0^2*m[[6]] - 8*y0*m[[7]] + m[[8]])
 )/(k^4*(-4*y0^2*m[[1]]^2 + 4*y0^2*m[[2]] + 4*y0*m[[1]]*m[[2]] - m[[2]]^2 -
        4*y0*m[[3]] + m[[4]])^2)
   ]


CDFQuadraticKurtosisExcess[cdf_, x_, y0_, k_:1, max_:1] :=
   CDFQuadraticKurtosis[cdf, x, y0, k, max] - 3


SYSInterquartileRange[sys_] := SYSQuantile[sys, 3/4] -
                               SYSQuantile[sys, 1/4]


CDFInterquartileRange[cdf_, x_, min_:0, max_:1] :=
    CDFQuantile[cdf, x, 3/4, $MachinePrecision, min, max] -
    CDFQuantile[cdf, x, 1/4, $MachinePrecision, min, max]


SYSQuartileDeviation[sys_] := SYSInterquartileRange[sys]/2


CDFQuartileDeviation[cdf_, x_, min_:0, max_:1] :=
    CDFInterquartileRange[cdf, x, min, max]/2


SYSPearsonSkewness2[sys_] :=
  3 (ExpectedState[sys] - SYSMedian[sys])/
     StateStandardDeviation[sys]


CDFPearsonSkewness2[cdf_, x_, min_:0, max_:1] :=
  3 (CDFExpectedState[cdf, x, max] - CDFMedian[cdf, x, min, max])/
     CDFStateStandardDeviation[cdf, x, max]


CDFRandom[cdf_, x_, n_:1, min_:0, max_:1] :=
    Table[CDFQuantileDown[cdf, x, Random[], $MachinePrecision,
          min, max], {n}]
```

J.6 StochasticAnalysis

```
perm2[p_List] := Module[{tab,size,m,n,a,i,j},
  n=Length[p];
  m=Table[Length[p[[i]]],{i,n}];
  size=Product[m[[i]],{i,n}];
  tab=Table[0,{size}];
  a=Table[1,{i,n}];
  Do[tab[[j]]=Table[p[[i,a[[i]]]],{i,n}];
     Do[If[Take[a,-(n-i)]==Take[m,-(n-i)],
           If[a[[i]]==m[[i]],a[[i]]=1,a[[i]]++]],{i,n-1}];
     If[a[[n]]==m[[n]],a[[n]]=1,a[[n]]++];
  ,{j,size}];
  tab]


systable2[p_List,phi_Symbol] := {#, phi[#]}& /@ perm2[p]


se2[sys_List] := Plus @@ ((#[[1]] #[[2]])& /@ sys)


FellerLists[n_Integer, l_Integer] := Module[{list,newlist},
   list=ReplacePart[Table[0,{l}],#,1]& /@ Range[n];
   Do[newlist={};
      Do[newlist=Join[newlist,Table[ReplacePart[list[[i]],j,k],
         {j,list[[i,k-1]]+1,n}]]
         ,{i,Length[list]}];
      list=newlist
   ,{k,2,l}];
   list]


ConsistentProbabilitiesQ[p_List, pprob_List] :=
  ((Length[#]& /@ p) == (Length[#]& /@ pprob)) &&
  (Table[1, {Length[p]}] == Apply[Plus, pprob, 1])


SystemFromDirectEnumeration[p_List,phi_Symbol,
      fphi_List,pprob_List] :=
  Module[{ans = Table[0, {Length[fphi]}], table, temp},
    table = systable2[p, phi];
    Do[temp=Transpose[Select[table, #[[2]]==fphi[[i]] &]][[1]];
       Do[Do[temp[[j,k]] =
             pprob[[k,Position[p[[k]], temp[[j,k]]][[1,1]]]]
         ,{k,Length[p]}]
```

```
       ,{j,Length[temp]}];
       ans[[i]] = Plus @@ Apply[Times, temp, 1]
    ,{i,Length[fphi]}];
    ans
  ]


SystemFromLBPInclusionExclusion[p_List,lbps_List,
       fphi_List,pprob_List] :=
   Module[{pprob1,lbps1,ans,clbp,v,fl,tmp},
      pprob1=Drop[FoldList[Plus,1,-#],-1]& /@ pprob;
      lbps1=Drop[#,-2]& /@ lbps;
      lbps1=MapAt[Position[fphi,#][[1,1]]&,#,2]& /@ lbps1;
      lbps1=MapAt[MapIndexed[Position[p[[#2[[1]]]],
         #1][[1,1]]&,#]&,#,1]& /@ lbps1;
      ans=Table[0,{Length[fphi]}]; ans[[1]]=1;
      Do[clbp=Transpose[Select[lbps1,#[[2]]==i &]][[1]];
         v=Length[clbp];
         Do[fl=FellerLists[v,j];
            tmp=Sum[Product[pprob1[[l,
               (Max /@ Transpose[clbp[[ fl[[k]] ]] ])[[l]] ]],
                {l,1,Length[p]}],{k,1,Length[fl]}];
            ans[[i]] += (-1)^(j+1)*tmp
         ,{j,1,v}]
      ,{i,2,Length[fphi]}];
      Append[Table[ans[[i-1]]-ans[[i]],
         {i,2,Length[ans]}],Last[ans]]]


TrivialBoundsFromLBP[p_List,lbps_List,pprob_List] :=
   Module[{pprob1},
      pprob1=Drop[FoldList[Plus,1,-#],-1]& /@ pprob;
      {Product[pprob1[[i,Position[ p[[i]],
         lbps[[i]] ][[1,1]] ]],{i,1,Length[pprob1]}],
       1-Product[1-pprob1[[i,Position[ p[[i]],
         lbps[[i]] ][[1,1]] ]],{i,1,Length[pprob1]}]
      }]


InclusionExclusionBoundsFromLBP[p_List,lbps_List,
       fphi_List,pprob_List,prec_Integer] :=
   Module[{pprob1,lbps1,ans,clbp,v,fl,tmp,templ,tempu,b},
      pprob1=Drop[FoldList[Plus,1,-#],-1]& /@ pprob;
      lbps1=Drop[#,-2]& /@ lbps;
      lbps1=MapAt[Position[fphi,
```

```
      #][[1,1]]&,#,2]& /@ lbps1;
    lbps1=MapAt[MapIndexed[Position[p[[#2[[1]]]],
      #1][[1,1]]&,#]&,#,1]& /@ lbps1;
    ans=Table[0,{Length[fphi]}]; ans[[1]]=1;
    Do[clbp=Transpose[Select[lbps1,#[[2]]==i &]][[1]];
      v=Length[clbp];
      b=Min[prec,v];
      If[b==v,
      Do[fl=FellerLists[v,j];
        tmp=Sum[Product[pprob1[[l,
          (Max /@ Transpose[clbp[[ fl[[k]] ]] ])[[l]] ]],
           {l,1,Length[p]}],{k,1,Length[fl]}];
        ans[[i]] += (-1)^(j+1)*tmp
      ,{j,1,v}],
      Do[fl=FellerLists[v,j];
        tmp=Sum[Product[pprob1[[l,
          (Max /@ Transpose[clbp[[ fl[[k]] ]] ])[[l]] ]],
           {l,1,Length[p]}],{k,1,Length[fl]}];
        ans[[i]] += (-1)^(j+1)*tmp
      ,{j,1,b-1}];
      templ=ans[[i]];
      fl=FellerLists[v,b];
        tmp=Sum[Product[pprob1[[l,
          (Max /@ Transpose[clbp[[ fl[[k]] ]] ])[[l]] ]],
           {l,1,Length[p]}],{k,1,Length[fl]}];
        tempu= templ+(-1)^(b+1)*tmp;
      If[templ<=tempu,ans[[i]]={templ,tempu},ans[[i]]={tempu,templ}]]
    ,{i,2,Length[fphi]}];
    ans]


SystemMatrix[fphi_List,ans_List] := Transpose[{fphi,ans}]


ReliabilityImportance[p_List, phi_Symbol, fphi_List, pprob_List,
       i_Integer, j_] :=
  Module[{high,low},
    high=ReplacePart[Table[0,{Length[p[[i]]]}],
      1, Position[p[[i]], j]];
    low=ReplacePart[Table[0,{Length[p[[i]]]}],
      1, 1];
    se2[SystemMatrix[fphi,
        SystemFromDirectEnumeration[p,phi,fphi,
          ReplacePart[pprob,high,i]]]]-
    se2[SystemMatrix[fphi,
```

```
        SystemFromDirectEnumeration[p,phi,fphi,
           ReplacePart[pprob,low,i]]]]
  ]


ReliabilityImportancesTable[p_List, phi_Symbol, fphi_List, pprob_List] :=
  Module[{ans},
    ans=p;
    Do[Do[ans[[i,j]]=
        ReliabilityImportance[p, phi, fphi, pprob, i, p[[i,j]]]
       ,{j,Length[p[[i]]]}]
    ,{i,Length[p]}];
  ans]


PToQ[p_List] := Drop[FoldList[Plus,1,-p],-1]


QToP[q_List] := Append[Table[q[[i-1]]-q[[i]],{i,2,Length[q]}],Last[q]]


StieltjesIntegral[f_, g_, {t_, min_:0, max_:1}, fact_:2] :=
   With[{di=(max-min)/(10^fact)},
      Sum[N[((g /. t->t0)-(g /. t->(t0-di))) (f /. t->t0)],
         {t0, min, max, di}]]


StieltjesIntegralH[f_, g_, {t_, min_:0, max_:1}, fact_:2] :=
   With[{di=(max-min)/(10^fact)},
      Sum[N[((g /. t->t0)-(g /. t->(t0-di))) (f /. t->t0)],
         {t0, min, max, di}] +
      NSum[((g /. t->t0)-(g /. t->(t0-di))) (f /. t->t0),
         {t0, max+di, Infinity, di}, Method->Fit]]


StieltjesIntegralG[f_, g_, {t_, min_:0, max_:1}, fact_:2] :=
   With[{di=(max-min)/(10^fact)},
      Sum[N[((g /. t->t0)-(g /. t->(t0-di))) (f /. t->t0)],
         {t0, min, max, di}] +
      NSum[((g /. t->t0)-(g /. t->(t0-di))) (f /. t->t0),
         {t0, max+di, Infinity, di}, Method->Fit] +
      NSum[((g /. t->t0)-(g /. t->(t0-di))) (f /. t->t0),
         {t0, min-di, -Infinity, -di}, Method->Fit]]


P[cdfs_List, rules___] := Times @@ (cdfs /. {rules})
```

```
S[cdfs_List, rules___] := 1-(Times @@ (1-(cdfs /. {rules})))

A[cdf_, x_, a_] := cdf /. x->x/a

C1[cdf1_, {cdf2_, min2_:0, max2_:1}, x_, z_, fact_:2] :=
   StieltjesIntegral[(cdf1 /. x->(z-x)), cdf2,
        {x, min2, max2}, fact] /. z->x

CA[cdf1_, min1_:0, max1_:1,
    {cdf2_, min2_:0, max2_:1}, x_, z_, fact_:2] :=
   A[C1[cdf1, {cdf2, min2, max2}, x, z, fact], x,
      1/(max1+max2)]

CAV[cdf1_, min1_:0, max1_:1,
    {cdf2_, min2_:0, max2_:1}, x_, z_, fact_:2] :=
   A[C1[cdf1, {cdf2, min2, max2}, x, z, fact], x, 1/2]

DiscreteBuild[syslist_, phi_] :=
  Module[{p,fphi,pprob,temp=syslist},
    Do[Do[temp[[i,j]]=syslist[[i,j,1]],
       {j,1,Length[syslist[[i]]]}],{i,Length[syslist]}]; p=temp;
    Do[Do[temp[[i,j]]=syslist[[i,j,2]],
       {j,1,Length[syslist[[i]]]}],{i,Length[syslist]}]; pprob=temp;
    fphi=Union[phi[#]& /@ perm2[p]];
    SystemMatrix[fphi,
       SystemFromDirectEnumeration[p, phi, fphi, pprob]]]

PM[syslist_] := Module[{phi},
   phi[x_] := Max[x]; DiscreteBuild[syslist,phi]]

SM[syslist_] := Module[{phi},
   phi[x_] := Min[x]; DiscreteBuild[syslist,phi]]

AM[sys_, a_] := Module[{ans=sys},
   Do[ans[[i,1]]=a sys[[i,1]],{i,Length[sys]}]; ans]

C1M[syslist_] := Module[{phi},
   phi[x_] := Plus @@ x; DiscreteBuild[syslist,phi]]

CAM[syslist_] := Module[{phi, ans, min, max},
   phi[x_] := Plus @@ x; ans=DiscreteBuild[syslist,phi];
   max=ans[[Length[ans],1]];
   AM[ans,1/max]]

CAVM[syslist_] := Module[{phi, ans},
   phi[x_] := Plus @@ x; ans=DiscreteBuild[syslist,phi];
   AM[ans,1/Length[syslist]]]
```

J.7 NewFunctions

The functions in this section require that all the other packages also be loaded.

```
phiround[x_,phi_,fphi_] := Module[{temp,temp2},
  temp=phi[x];
  temp2=Min[(fphi-temp)^2];
  First[Select[fphi,((#-temp)^2 // N) == (temp2 // N)&,1]]]


Bounds2[data_List,f_List,ag_Integer:6] :=
Module[{a,b,ints,mid,total={0,0}},
a=ints=Union /@ Transpose[Transpose[data][[1]]];
b=Apply[List,Flatten[Array[Unique[],(Length /@ a)-1,2]],1];
Do[If[ag>0,ints[[i,j]] = NIntegrate[f[[i]],
           {x,a[[i,j-1]],a[[i,j]]},AccuracyGoal->ag],
      ints[[i,j]] = (f[[i]] /. x->a[[i,j]])-
           (f[[i]] /. x->a[[i,j-1]]) // N]
,{i,Length[a]},{j,2,Length[a[[i]]]}];
Do[mid = ((#[[1]]+#[[2]])/2)& /@ Table[{a[[i,b[[j,i]]-1]],
         a[[i,b[[j,i]]]]},{i,Length[a]}];
   total += {PhiMin[mid,data],PhiMax[mid,data]}*
            Product[ints[[i,b[[j,i]]]],{i,Length[a]}]
,{j,Length[b]}];
total]


PhiMax[x_List, data_List] := Min[Transpose[Select[
   data,FreeQ[Inner[GreaterEqual,#[[1]],x,List],
   False]&]][[2]]]


PhiMin[x_List, data_List] := Max[Transpose[Select[
   data,FreeQ[Inner[LessEqual,#[[1]],x,List],
   False]&]][[2]]]


SystemFromDirectEnumerationLow[p_List,data_List,
      fphi_List,pprob_List] :=
  Module[{ans = Table[0, {Length[fphi]}], table, temp},
    table = systable2low[p, phi, data];
    Do[temp=Transpose[Select[table, #[[2]]==fphi[[i]] &]][[1]];
       Do[Do[temp[[j,k]] =
             pprob[[k,Position[p[[k]], temp[[j,k]]][[1,1]]]]
          ,{k,Length[p]}]
```

```
       ,{j,Length[temp]}];
       ans[[i]] = Plus @@ Apply[Times, temp, 1]
    ,{i,Length[fphi]}];
    ans
  ]


systable2low[p_List,phi_Symbol,data_List] :=
   {#, PhiMin[#,data]}& /@ perm2[p]


SystemFromDirectEnumerationHigh[p_List,data_List,
      fphi_List,pprob_List] :=
  Module[{ans = Table[0, {Length[fphi]}], table, temp},
    table = systable2high[p, phi, data];
    Do[temp=Transpose[Select[table, #[[2]]==fphi[[i]] &]][[1]];
       Do[Do[temp[[j,k]] =
             pprob[[k,Position[p[[k]], temp[[j,k]]][[1,1]]]]
          ,{k,Length[p]}]
       ,{j,Length[temp]}];
       ans[[i]] = Plus @@ Apply[Times, temp, 1]
    ,{i,Length[fphi]}];
    ans
  ]


systable2high[p_List,phi_Symbol,data_List] :=
   {#, PhiMax[#,data]}& /@ perm2[p]


MultiQuadricND2[inpdata_,rsq_:(1/6),
   prec_:$MachinePrecision] :=
Module[{data,grid,obt,n,temp,tempc,low,high,ans,a1,a0,a},
(* Initialize Main Variables *)
   data=ExtremaAdd[inpdata];
   n=Length[data[[1,1]]];
(* Create Grid *)
   a=Union /@ Transpose[Transpose[data][[1]]];
   grid = Table[a[[i,#1[[i]]+1]],{i,n}]& /@
     Apply[List,Flatten[Array[Unique[],Length /@ a,0]],2];
   grid=Transpose[Sort[Transpose[{grid,Apply[Plus,
     ((grid-1/2)^2),2]}],
     OrderedQ[{#1[[2]],#2[[2]]}]&]][[1]];
(* Calculate Grid Points Known Based on Monotonicity *)
   a1=Transpose[Select[data,Chop[(1-#[[2]])]==0&]][[1]];
```

```
   a0=Transpose[Select[data,Chop[#[[2]]]==0&]][[1]];
   If[Length[a1]>1 || Length[a0]>1,
     Do[Do[If[LessQ[grid[[i]],a0[[j]]],
           If[FreeQ[Transpose[data][[1]],grid[[i]]],
              data=Prepend[data,{grid[[i]],0}]]]
       ,{j,Length[a0]}];
       Do[If[GreaterQ[grid[[i]],a1[[j]]],
          If[FreeQ[Transpose[data][[1]],grid[[i]]],
             data=Prepend[data,{grid[[i]],1}]]]
       ,{j,Length[a1]}];
     ,{i,Length[grid]}]];
(* Calculated C Matrix, Based on Known Values *)
   tempc=MultiQuadricC[data,rsq,prec];
(* Calculate Answers *)
   obt=data;
   ans=Table[1/2,{Length[grid]}]; (* assumes Mi=1/2 *)
   Print["Calculation is Complete When Zero is Reached"];
   Do[Print[Length[grid]-i];
      If[FreeQ[Transpose[data][[1]],grid[[i]]],
         low=0; high=1;
         temp=MultiQuadric[grid[[i]],data,tempc,rsq,prec];
         Do[If[LessQ[obt[[j,1]],grid[[i]]],
               low=Max[low,obt[[j,2]]]];
            If[GreaterQ[obt[[j,1]],grid[[i]]],
               high=Min[high,obt[[j,2]]]];
         ,{j,Length[obt]}];
         If[temp<low,temp=low]; If[temp>high,temp=high];
         obt=Prepend[obt,{grid[[i]],temp}];
         ans[[i]]=temp,
      ans[[i]]=Select[data,#[[1]]==grid[[i]]&][[1,2]]]
   ,{i,Length[grid]}];
(* Return Final Answer *)
Transpose[{grid,ans}]]


MultiQuadricNDRMSError[data_,phi_,m_:10] :=
  Module[{real2,n}, n=Length[data[[1,1]]];
    real2=PhiGrid[phi,m,n];
    Sqrt[Sum[(real2[[i,2]] -
           MLinInt[real2[[i,1]], data])^2,
       {i, Length[real2]}]/Length[real2]]]


PhiMinRMSError[data_,phi_,m_:10,
   prec_:$MachinePrecision] :=
```

```
Module[{real,n},
   n=Length[data[[1,1]]];
   real=PhiGrid[phi,m,n];
   Sqrt[Sum[(real[[i,2]]-PhiMin[real[[i,1]],data])^2
   ,{i,Length[real]}]/Length[real]]]


PhiMaxRMSError[data_,phi_,m_:10,
   prec_:$MachinePrecision] :=
Module[{real,n},
   n=Length[data[[1,1]]];
   real=PhiGrid[phi,m,n];
   Sqrt[Sum[(real[[i,2]]-PhiMax[real[[i,1]],data])^2
   ,{i,Length[real]}]/Length[real]]]


MuSigma[inp_List,expr_] :=
  Module[{mean=expr, tol=expr},
  tol=Sqrt[Sum[D[tol,inp[[i,1]]]^2*inp[[i,3]]^2,
              {i,Length[inp]}]];
  Do[{mean,tol}={mean,tol} /.
    {inp[[i,1]] -> inp[[i,2]]},{i,Length[inp]}];
  N[{mean,tol}]]


SVStillOut[mmax_]:=Module[{t},
  Table[{c,t/.FindRoot[Evaluate[D[StateVariance[
  Transpose[{Table[i/c,{i,0,c}],
    PDPErlangian[c,1,t]}]],t]],
    {t,c*3/5}]},{c,1,mmax}]]
```