


INTERNATIONAL SERIES IN OPERATIONS
RESEARCH AND MANAGEMENT SCIENCE



Computational Probability

Algorithms and Applications in the
Mathematical Sciences

John H. Drew
Diane L. Evans
Andrew G. Glen
Lawrence M. Leemis

 Springer

Computational Probability

**Recent titles in the INTERNATIONAL SERIES
IN OPERATIONS RESEARCH & MANAGEMENT SCIENCE**
Frederick S. Hillier, Series Editor, Stanford University

Sethi, Yan & Zhang/*INVENTORY AND SUPPLY CHAIN MANAGEMENT WITH FORECAST
UPDATES*

Cox/*QUANTITATIVE HEALTH RISK ANALYSIS METHODS: Modeling the Human Health Impacts
of Antibiotics Used in Food Animals*

Ching & Ng/*MARKOV CHAINS: Models, Algorithms and Applications*

Li & Sun/*NONLINEAR INTEGER PROGRAMMING*

Kaliszewski/*SOFT COMPUTING FOR COMPLEX MULTIPLE CRITERIA DECISION MAKING*

Bouyssou et al./*EVALUATION AND DECISION MODELS WITH MULTIPLE CRITERIA: Stepping
stones for the analyst*

Blecker & Friedrich/*MASS CUSTOMIZATION: Challenges and Solutions*

Appa, Pitsoulis & Williams/*HANDBOOK ON MODELLING FOR DISCRETE OPTIMIZATION*

Herrmann/*HANDBOOK OF PRODUCTION SCHEDULING*

Axsäter/*INVENTORY CONTROL, 2nd Ed.*

Hall/*PATIENT FLOW: Reducing Delay in Healthcare Delivery*

Józefowska & Węglarz/*PERSPECTIVES IN MODERN PROJECT SCHEDULING*

Tian & Zhang/*VACATION QUEUEING MODELS: Theory and Applications*

Yan, Yin & Zhang/*STOCHASTIC PROCESSES, OPTIMIZATION, AND CONTROL THEORY
APPLICATIONS IN FINANCIAL ENGINEERING, QUEUEING NETWORKS,
AND MANUFACTURING SYSTEMS*

Saaty & Vargas/*DECISION MAKING WITH THE ANALYTIC NETWORK PROCESS: Economic,
Political, Social & Technological Applications w. Benefits, Opportunities, Costs & Risks*

Yu/*TECHNOLOGY PORTFOLIO PLANNING AND MANAGEMENT: Practical Concepts and Tools*

Kandiller/*PRINCIPLES OF MATHEMATICS IN OPERATIONS RESEARCH*

Lee & Lee/*BUILDING SUPPLY CHAIN EXCELLENCE IN EMERGING ECONOMIES*

Weintraub/*MANAGEMENT OF NATURAL RESOURCES: A Handbook of Operations Research
Models, Algorithms, and Implementations*

Hooker/*INTEGRATED METHODS FOR OPTIMIZATION*

Dawande et al./*THROUGHPUT OPTIMIZATION IN ROBOTIC CELLS*

Friesz/*NETWORK SCIENCE, NONLINEAR SCIENCE and INFRASTRUCTURE SYSTEMS*

Cai, Sha & Wong/*TIME-VARYING NETWORK OPTIMIZATION*

Mamon & Elliott/*HIDDEN MARKOV MODELS IN FINANCE*

del Castillo/*PROCESS OPTIMIZATION: A Statistical Approach*

Józefowska/*JUST-IN-TIME SCHEDULING: Models & Algorithms for Computer & Manufacturing
Systems*

Yu, Wang & Lai/*FOREIGN-EXCHANGE-RATE FORECASTING WITH ARTIFICIAL NEURAL
NETWORKS*

Beyer et al./*MARKOVIAN DEMAND INVENTORY MODELS*

Shi & Olafsson/*NESTED PARTITIONS OPTIMIZATION: Methodology And Applications*

Samaniego/*SYSTEM SIGNATURES AND THEIR APPLICATIONS IN ENGINEERING RELIABILITY*

Kleijnen/*DESIGN AND ANALYSIS OF SIMULATION EXPERIMENTS*

Førsund/*HYDROPOWER ECONOMICS*

Kogan & Tapiero/*SUPPLY CHAIN GAMES: Operations Management and Risk Valuation*

Vanderbei/*LINEAR PROGRAMMING: Foundations & Extensions, 3rd Edition*

Chhajed & Lowe/*BUILDING INTUITION: Insights from Basic Operations Mgmt. Models
and Principles*

Luenberger & Ye/*LINEAR AND NONLINEAR PROGRAMMING, 3rd Edition*

*** A list of the early publications in the series is at the end of the book ***

John H. Drew

Diane L. Evans

Andrew G. Glen

Lawrence M. Leemis

Computational Probability

Algorithms and Applications in the
Mathematical Sciences



Springer

John H. Drew
College of William and Mary
Williamsburg, VA, USA

Diane L. Evans
Rose-Hulman Institute of Technology
Terre Haute, IN, USA

Andrew G. Glen
United States Military Academy
West Point, NY, USA

Lawrence M. Leemis
College of William and Mary
Williamsburg, VA, USA

Series Editor:

Fred Hillier
Stanford University
Stanford, CA, USA

ISBN 978-0-387-74675-3

e-ISBN 978-0-387-74676-0

Library of Congress Control Number: 2007933820

© 2008 Springer Science+Business Media, LLC

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper.

9 8 7 6 5 4 3 2 1

springer.com

Preface

For decades, statisticians have enjoyed the use of “statistical packages” which read in a (potentially) large data set, process the observations, and print out anything from histograms to sample variances, to p -values, to multi-dimensional plots. But pity the poor probabilist, who through all those decades had only paper and pencil for symbolic calculations. The purpose of this monograph is to address the plight of the probabilist by providing algorithms to perform calculations associated with univariate random variables. We refer to a collection of data structures and algorithms that automate probability calculations as “computational probability.” The data structures and algorithms introduced here have been implemented in a language known as APPL (A Probability Programming Language). Several illustrations of problems from the mathematical sciences that can be solved by implementing these algorithms in a computer algebra system are presented in the final chapters of this monograph.

The algorithms for manipulating random variables (e.g., adding, multiplying, transforming, ordering) symbolically result in an entire class of new problems that can now be addressed. The implementation of these algorithms in Maple-based APPL is available without charge for non-commercial use at www.applsoftware.com. APPL is able to perform exact probability calculations for problems that would otherwise be deemed intractable. The work is quite distinct from traditional probability analysis in that a computer algebra system, in this case Maple, is used as a computing platform.

The use of a computer algebra system to solve problems in operations research and probability is increasing. Other researchers also sense the benefits of incorporating a computer algebra system into fields with probabilistic applications, for example, Parlar’s *Interactive Operations Research with Maple* [69], Karian and Tanis’s 2nd edition of *Probability and Statistics: Explorations with Maple* [42], Rose and Smith’s *Mathematical Statistics and Mathematica* [74], and Hasting’s 2nd edition of *Introduction to the Mathematics of Operations Research with Mathematica* [33].

This monograph significantly differs from the four titles listed above in two ways. First, the four titles listed above are all textbooks, rather than research monographs. They contain exercises and examples geared toward students, rather than researchers. Second, the emphasis in most of these texts is much broader than the emphasis being proposed here. For example, Parlar and Hasting consider all of OR/MS, rather than the probabilistic side of OR/MS proposed here in much more depth. Also, Karian and Tanis emphasize Monte Carlo solutions to probability and statistics problems, as opposed to the exact solutions given in APPL.

The monograph begins with an introductory chapter, then in Chapter 2 reviews the Maple data structures and functions necessary to implement APPL. This is followed by a discussion of the development of the algorithms (Chapters 3–5 for continuous random variables and Chapters 6–8 for discrete random variables), and by a sampling of various applications in the mathematical sciences (Chapters 9–11). The two most likely audiences for the monograph are researchers in the mathematical sciences with an interest in applied probability and instructors using the monograph for a special topics course in computational probability taught in a mathematics, statistics, operations research, management science, or industrial engineering department. The intended audience for this monograph includes researchers, MS students, PhD students, and advanced practitioners in stochastic operations research, management science, and applied probability.

An indication of the proven utility of APPL is that the research efforts of the authors and other colleagues have produced many related refereed journal publications, many conference presentations, the ICS Computing Prize with INFORMS, a government patent, and multiple improvements to pedagogical methods in numerous colleges and universities around the world. We believe that the potential of this field of computational probability in research and education is unlimited. It is our hope that this monograph encourages people to join us in attaining future accomplishments in this field.

We are grateful to Camille Price and Gary Folven from Springer for their support of this project. We also thank Professor Ludolf Meester from TU Delft for class-testing and debugging portions of the APPL code. We thank our co-authors Matt Duggan, Kerry Connell, Jeff Mallozzi, and Bruce Schmeiser for their collaboration on the applications given in Chapter 11. We also thank the editors, anonymous referees, and colleagues who have been helpful in the presentation of the material, including John Backes, Donald Barr, Barbara Boyer, Don Campbell, Jacques Carette, Gianfranco Ciardo, Mike Crawford, Mark Eaton, Jerry Ellis, Bob Foote, Greg Gruver, Matt Hanson, Carl Harris, Billy Kaczynski, Rex Kincaid, Hank Krieger, Marina Kondratovitch, Sid Lawrence, Lee McDaniel, Lauren Merrill, David Nicol, Raghu Pasupathy, Steve Roberts, Evan Saltzman, Jim Scott, Bob Shumaker, Paul Stockmeyer, Bill Treadwell, Michael Trosset, Erik Vargo, Mark Vinson, and Marianna Williamson. We thank techie Robert Marmorstein for his \LaTeX support. The authors gratefully acknowledge support from the Clare Boothe

Luce Foundation, the National Science Foundation (for providing funding for an Educational Innovation Grant CDA9712718 “Undergraduate Modeling and Simulation Analysis” and for scholarship funds provided under the CSEMS Grant 0123022 “Effective Transitions Through Academe to Industry for Computer Scientists and Mathematicians”), and the College of William & Mary for research leave to support this endeavor.

Williamsburg, VA
Terre Haute, IN
West Point, NY
Williamsburg, VA

John Drew
Diane Evans
Andy Glen
Larry Leemis

NOTE

The authors and publisher of this book have made their best effort in preparing this book and the associated software. The authors and publisher of this book make no warranty of any kind, expressed or implied, with respect to the software or the associated documentation described in this book. Neither the authors nor the publisher shall be liable in any event for incidental or consequential damages in connection with, or arising out of, the furnishing, performance, or use of these programs or the associated documentation described in this book.

Contents

Part I Introduction

1	Computational Probability	3
1.1	Four Simple Examples of the Use of APPL	3
1.2	A Different Way of Thinking	8
1.3	Overview	10
2	Maple for APPL	13
2.1	Numerical Computations	13
2.2	Variables	15
2.3	Symbolic Computations	16
2.4	Functions	17
2.5	Data Types	18
2.6	Solving Equations	20
2.7	Graphing	22
2.8	Calculus	24
2.9	Loops and Conditions	26
2.10	Procedures	27

Part II Algorithms for Continuous Random Variables

3	Data Structures and Simple Algorithms	33
3.1	Data Structures	33
3.2	Simple Algorithms	37
4	Transformations of Random Variables	45
4.1	Theorem	46
4.2	Implementation in APPL	48
4.3	Examples	50

5	Products of Random Variables	55
5.1	Theorem	56
5.2	Implementation in APPL	58
5.3	Examples	60
5.4	Extensions	64
5.5	Algorithm	65

Part III Algorithms for Discrete Random Variables

6	Data Structures and Simple Algorithms	71
6.1	Data Structures	71
6.2	Simple Algorithms	81
7	Sums of Independent Random Variables	91
7.1	Preliminary Examples	91
7.2	Conceptual Algorithm Development	95
7.3	Algorithm	106
7.4	Implementation Issues	108
7.5	Examples	110
8	Order Statistics	119
8.1	Notation and Taxonomy	119
8.2	Sampling Without Replacement	121
8.3	Sampling With Replacement	126
8.4	Extension	130

Part IV Applications

9	Reliability and Survival Analysis	135
9.1	Systems Analysis	135
9.2	Lower Confidence Bound on System Reliability	140
9.3	Survival Analysis	144
10	Stochastic Simulation	153
10.1	Tests of Randomness	153
10.2	Input Modeling	159
10.3	Kolmogorov–Smirnov Goodness-of-Fit Test	167
11	Other Applications	185
11.1	Stochastic Activity Networks	185
11.2	Benford’s Law	194
11.3	Miscellaneous Applications	199

References	207
-------------------------	-----

Index	213
--------------------	-----

Computational Probability

The purpose of this chapter is to get you to read the rest of the monograph. We present four examples of probability questions that would be unpleasant to solve by hand, but are solvable with computational probability using A Probability Programming Language (APPL). We define the field of *computational probability* as the development of data structures and algorithms to automate the derivation of existing and new results in probability and statistics. Section 10.3, for example, contains the derivation of the distribution of a well-known test statistic that requires 99500 carefully crafted integrations.

1.1 Four Simple Examples of the Use of APPL

The first example is one that a probability student would solve approximately by using the central limit theorem or Monte Carlo simulation. We show that APPL can compute the desired result exactly.

Example 1.1. Let X_1, X_2, \dots, X_{10} be independent and identically distributed (iid) random variables that are uniformly distributed between 0 and 1. Find the probability that their sum lies between 4 and 6, i.e.,

$$\Pr \left(4 < \sum_{i=1}^{10} X_i < 6 \right).$$

The central limit theorem yields only one digit of accuracy for this particular problem. Monte Carlo requires custom programming and the result is typically stated as an interval around the true value. Also, the number of replications required is a quadratic function of the desired accuracy: each additional digit of accuracy requires a 100-fold increase in the number of Monte Carlo replications. On the other hand, the APPL statements to solve this problem are

```

> X := UniformRV(0, 1);
> Y := ConvolutionIID(X, 10);
> CDF(Y, 6) - CDF(Y, 4);

```

which yields

$$\Pr\left(4 < \sum_{i=1}^{10} X_i < 6\right) = \frac{655177}{907200} \cong 0.7222.$$

The first line of APPL code defines X as a $U(0, 1)$ random variable. The second line defines the random variable Y as the sum of 10 iid random variables, each having the same distribution as X . Finally, the last line evaluates the cumulative distribution function (CDF) of Y at 6 less the CDF of Y at 4, yielding the exact result as the fraction shown above.

The second example concerns finding the probability density function (PDF) of the product of two triangular random variables.

Example 1.2. Let X and Y be independent random variables with triangular distributions with parameters $(1, 2, 4)$ and $(1, 2, 3)$, respectively. (The three parameters are the minimum, mode, and maximum, respectively.) Find and plot the PDF of $V = XY$.

The APPL code to solve this problem is:

```

> X := TriangularRV(1, 2, 4);
> Y := TriangularRV(1, 2, 3);
> V := Product(X, Y);
> PlotDist(V);

```

which returns the PDF of V as

$$f_V(v) = \begin{cases} -\frac{4}{3}v + \frac{2}{3}\ln v + \frac{2v}{3}\ln v + \frac{4}{3} & 1 < v \leq 2 \\ -8 + \frac{14}{3}\ln 2 + \frac{7v}{3}\ln 2 + \frac{10}{3}v & \\ \quad -4\ln v - \frac{5v}{3}\ln v & 2 < v \leq 3 \\ -4 + \frac{14}{3}\ln 2 + \frac{7v}{3}\ln 2 + 2v & \\ \quad -2\ln v - v\ln v - 2\ln 3 - \frac{2v}{3}\ln 3 & 3 < v \leq 4 \\ \frac{44}{3} - 14\ln 2 - \frac{7v}{3}\ln 2 - \frac{8}{3}v & \\ \quad -2\ln 3 + \frac{22}{3}\ln v - \frac{2v}{3}\ln 3 + \frac{4v}{3}\ln v & 4 < v \leq 6 \\ \frac{8}{3} - 8\ln 2 - \frac{4v}{3}\ln 2 - \frac{2}{3}v & \\ \quad + \frac{4}{3}\ln v + \frac{v}{3}\ln v + 4\ln 3 + \frac{v}{3}\ln 3 & 6 < v \leq 8 \\ -8 + 8\ln 2 + \frac{2v}{3}\ln 2 + \frac{2}{3}v & \\ \quad + 4\ln 3 - 4\ln v + \frac{v}{3}\ln 3 - \frac{v}{3}\ln v & 8 < v < 12. \end{cases}$$

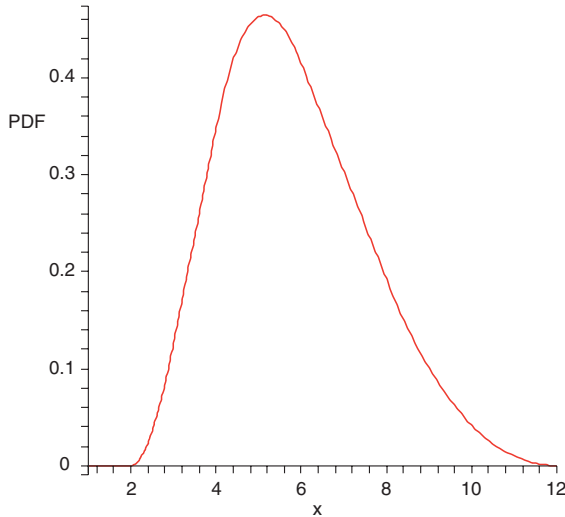


Fig. 1.1. PDF of $V = XY$.

The first two APPL statements define X and Y as triangular random variables. The third statement uses the APPL **Product** procedure to compute the PDF of the product. The last statement plots the PDF of V , which is shown in Figure 1.1.

The first two examples are complex enough to require about an hour to work out by hand, assuming that no mathematical or calculation errors occur. In both examples, the result was available with just a few lines of code and a few seconds of computer time. We next turn to a problem involving discrete random variables.

Example 1.3. A bag contains 15 billiard balls, numbered 1 to 15. If 7 balls are drawn from the bag at random, find the probability that the median number drawn is 5 when (a) sampling is performed without replacement; (b) sampling is performed with replacement.

In both of the solutions given below, the discrete random variable Y denotes the median number drawn in the sample of size 7. In part (a), the support of Y is $4, 5, \dots, 12$ due to the sampling without replacement, but in part (b), the support of Y is $1, 2, \dots, 15$ due to the fact that each ball drawn is placed back into the bag after being sampled.

(a) The APPL code in the sampling without replacement case is

```
> X := UniformDiscreteRV(1, 15);
> Y := OrderStat(X, 7, 4, "wo");
> PDF(Y, 5);
```

which returns the probability that the median is 5 as

$$\Pr(Y = 5) = \frac{32}{429} \cong 0.07459.$$

The first APPL statement establishes that the distribution X has the uniform discrete distribution (or rectangular distribution), which implies that each of the balls in the bag is equally likely to be drawn. The second statement makes a call to the APPL procedure **OrderStat** with four arguments: the population distribution, the number of draws from the population, the order statistic of interest, and the optional "wo" argument to indicate that sampling is without replacement (the default is with replacement). Finally, the third statement evaluates the PDF of Y at 5, giving the desired result.

(b) The APPL code in the sampling with replacement case is

```
> X := UniformDiscreteRV(1, 15);
> Y := OrderStat(X, 7, 4);
> PDF(Y, 5);
```

which returns the probability that the median is 5 as

$$\Pr(Y = 5) = \frac{2949971}{34171875} \cong 0.08633.$$

The APPL code is identical to the without replacement case except that the optional fourth argument in **OrderStat** has been defaulted.

Although these two quantities of interest could have been calculated by hand, the calculations would have been much more tedious if some of the billiard balls were more likely to be drawn than others. Such a change, however, would not pose a difficulty for APPL. Also, although it has not been exploited in the two calls to **OrderStat**, the procedure has computed the entire distribution of Y , so expected values and other subsequent computations on Y could be performed.

The three examples given thus far have concerned well-known distributions (i.e., triangular, uniform) parameterized by constants. The fourth and final example, drawn from a popular mathematical statistics text, highlights a non-standard distribution with symbolic parameters. Unlike most of the earlier examples, this example can be worked by hand, then the solution can be checked using APPL. Furthermore, a student can change any of the arguments in the problem (e.g., the sample size or critical value) to see if the results of the change match his or her intuition.

Example 1.4. Let X_1 and X_2 be iid observations drawn from a population with PDF

$$f(x) = \theta x^{\theta-1} \quad 0 < x < 1,$$

where $\theta > 0$. Test $H_0: \theta = 1$ versus $H_1: \theta > 1$ using the test statistic X_1X_2 and the critical region $C = \{(X_1, X_2) \mid X_1X_2 \geq 3/4\}$. Find the power function and significance level α for the test (Hogg, et al. [37, page 270]).

The APPL code to compute the power function is

```
> n := 2;
> c := 3 / 4;
> assume(theta > 0);
> X := [[x -> theta * x ^ (theta - 1)], [0, 1],
        ["Continuous", "PDF"]];
> T := ProductIID(X, n);
> power := SF(T, c);
```

which yields

$$\Pr(\text{rejecting } H_0 \mid \theta) = 1 - (3/4)^\theta + \theta(3/4)^{\theta-1} \ln(3/4)$$

for $\theta > 0$. The sample size n is defined in the first APPL statement. The critical value c is defined next. The Maple `assume` procedure defines the parameter space $\theta > 0$. The fact that the population distribution is non-standard requires the random variable X to be defined using the “list-of-sublists” data structure used in APPL (described subsequently in Chapters 3 and 6). The first sublist gives the functional form of the PDF, the second sublist gives the support, and the third sublist indicates that the random variable being defined is continuous and that the function in the first sublist is a PDF. The `ProductIID` procedure computes the product of the two iid random variables having the distribution of X and creates the random variable T . Finally, the power function is defined using the survivor function (`SF`) procedure to compute $\Pr(T \geq c = 3/4)$ for various values of $\theta > 0$.

To compute the significance level of the test, the additional Maple statement

```
> alpha := subs(theta = 1, power);
```

is required, yielding

$$\alpha = 1/4 + (3/4) \ln(3/4) \cong 0.0342.$$

Plotting the power function requires the additional Maple statement

```
> plot(power, theta = 0 .. 20);
```

The power function is shown in Figure 1.2. Obviously, this example can be generalized for different sample sizes, population distributions, and critical values with only minor modifications.

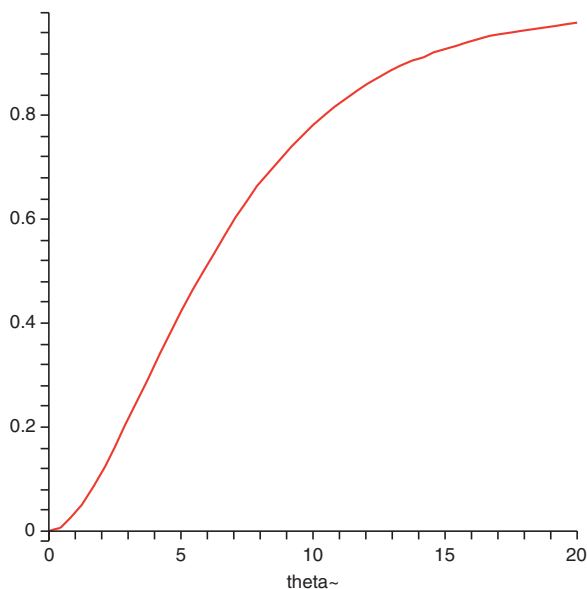


Fig. 1.2. The power function.

The careful reader has probably noted the distinction between APPL statements and Maple statements from the previous example. If APPL is to be a serious research tool, then it is necessary to learn certain aspects of Maple, such as the `subs` and `plot` procedures used in the fourth example. A brief review of those aspects of Maple that are germane to APPL are given in Chapter 2.

The next section shows that there is a non-traditional way of thinking that is required in order to mix probability and computing.

1.2 A Different Way of Thinking

A few weeks into the introductory calculus-based probability class, the properties associated with the PDF of a continuous random variable

$$\int_{-\infty}^{\infty} f(x) dx = 1 \quad \text{and} \quad f(x) \geq 0, -\infty < x < \infty$$

are introduced. Probability definitions and problems often need to be recast in order to make them amenable to processing by a computer algebra system. The first property is fairly easy to verify in a computer algebra system [if $f(x)$ can be integrated symbolically, do so and show that the area under the curve is 1, else, integrate numerically and show that the area under the curve is within epsilon of 1]. The second property, however, is quite difficult to show.

One might imagine checking the PDF on a very fine grid over the support of the random variable, but this is unappealing for two reasons. First, the grid should be very fine, resulting in considerable CPU time. Second, and more importantly, if $f(x)$ drops below the x -axis between grid points, a PDF will be accepted with $f(x) < 0$.

A better way to verify the PDF properties is to recast them as the equivalent pair

$$\int_{-\infty}^{\infty} f(x) dx = 1 \quad \text{and} \quad \int_{-\infty}^{\infty} |f(x)| dx = 1,$$

which can be easily verified in a computer algebra system. An APPL procedure named **VerifyPDF** verifies this latter pair of properties for a continuous random variable rather than the more intuitive pair typically introduced in an introductory probability class.

As a second example of a more algorithmic approach to thinking about probability, consider the following problem. If $X \sim U(0, 1)$, find the distribution of $Y = g(X) = X^2$. The “transformation technique” is commonly used to find the distribution of Y when $g(X)$ is a 1–1 transformation from the support of X , which we call A , to the support of Y , which we call B . The general formula for the PDF of Y is

$$f_Y(y) = f_X(g^{-1}(y)) \left| \frac{dg^{-1}(y)}{dy} \right| \quad y \in B.$$

Since the PDF of X is $f_X(x) = 1$ for $0 < x < 1$ and the inverse transformation is $g^{-1}(y) = \sqrt{y}$, the PDF of Y is

$$f_Y(y) = f_X(\sqrt{y}) \left| \frac{1}{2\sqrt{y}} \right| \quad 0 < y < 1$$

or

$$f_Y(y) = \frac{1}{2\sqrt{y}} \quad 0 < y < 1.$$

This is exactly the way that the transformation technique should be implemented in an algorithm. In fact, the APPL code to find the distribution of Y is

```
> X := UniformRV(0, 1);
> g := [[x -> x ^ 2], [0, 1]];
> Y := Transform(X, g);
```

which returns the PDF of Y as

$$f_Y(y) = \frac{1}{2\sqrt{y}} \quad 0 < y < 1$$

as expected. Unfortunately, the algorithm behind the **Transform** procedure is not quite as simple as this example would suggest. For example, when Maple

finds the inverse function $g^{-1}(y)$ it returns not the single inverse suggested above, but rather two inverses:

$$g_1^{-1}(y) = +\sqrt{y} \quad \text{and} \quad g_2^{-1}(y) = -\sqrt{y}.$$

How do we determine which one is the correct inverse? Here is the thinking: First we determine the midpoint of the support of X , which we denote by x_0 . In the current example, $x_0 = 1/2$. Next we calculate the value of the transformation at x_0 , which is $g(x_0) = 1/4$. Finally, we loop through all of the inverses that are returned by the computer algebra system inserting $g(x_0)$ as an argument, and determine which of the inverses returns x_0 . In this case, $g_1^{-1}(g(x_0)) = x_0$, so the first inverse is the right one and the second inverse should be discarded. Adjustments in this procedure must be made for distributions with infinite support, such as the exponential distribution and the normal distribution.

Likewise, what if the distribution of X is defined in a piecewise fashion, e.g., the triangular distribution? In this case an outside loop must be added to the **Transform** procedure in order to do the appropriate bookkeeping so that all of the density gets transformed from the distribution of X to the distribution of Y appropriately.

Furthermore, how does the **Transform** procedure handle portions of the transformation that are 2-1, or, more generally, k -1? The algorithm behind the **Transform** procedure tracks each piecewise segment and determines the associated mapping onto the support of Y .

These two examples provide a window into the thinking that is required to develop algorithms for performing probability calculations. APPL requires many conditions [e.g., if $g_1^{-1}(g(x_0)) = x_0$, then choose the right inverse] and loops [e.g., loop through all inverses] to perform various manipulations of random variables. Automating probability calculations in this manner is precisely the vision we have for computational probability. APPL and similar languages can be used to explore new probability theory or calculate known but mathematically intractable probability measures.

1.3 Overview

We end this chapter with an overview of the organization of the monograph. As mentioned earlier, Chapter 2 contains a brief review of Maple syntax, data structures, and programming constructs used to write the procedures that comprise APPL. We survey only a small portion of the Maple language.

The second part of the monograph, Chapters 3-5, considers continuous random variables. The data structure used for defining a continuous random variable is defined in Chapter 3. Chapters 4 and 5 contain examples of algorithms devised for manipulating continuous random variables. Chapter 4 considers transformations of continuous random variables and Chapter 5 considers products of continuous random variables.

The third part of the monograph, Chapters 6–8, considers discrete random variables. The data structure that we have used for defining a discrete random variable is defined in Chapter 6. Chapters 7 and 8 contain examples of algorithms for manipulating discrete random variables. Chapter 7 considers sums of discrete random variables and Chapter 8 considers the distribution of order statistics drawn from discrete distributions.

The fourth part of the monograph, Chapters 9–11, considers applications of APPL in computational probability. Chapter 9 contains applications in reliability and survival analysis problems, including system design, lower confidence bounds on system reliability, and bootstrapping. Chapter 10 contains APPL applications in discrete-event simulation, including random number testing, input modeling, and goodness-of-fit testing. Finally, Chapter 11 contains miscellaneous applications, such as determining the exact distribution of the time to complete a stochastic activity network, probabilistic analysis of Benford’s law, and the generation of values in statistical tables.

Maple for APPL

Maple is a computer algebra system and programming language that can be used for numerical computations, solving equations, manipulating symbolic expressions, plotting, and programming, just to name a few of the basics. APPL is, simply, a set of supplementary Maple commands and procedures that augments the existing computer algebra system. In effect, APPL takes the capabilities of Maple and turns it into a computer algebra system for computational probability. This chapter contains guidelines for using Maple, and it discusses the Maple commands that are used in APPL programming. Upon reading this chapter, an APPL user gains the knowledge necessary to modify the APPL code to meet his or her particular needs. We will start with a discussion of basic numeric computation, then advance to defining variables, symbolic computations, functions, data types, solving equations, calculus and graphing. Then we will discuss the programming features of Maple that facilitate building the APPL language: loops, conditions and procedures.

2.1 Numerical Computations

Numerical computations in Maple give it the functionality of a hand-held calculator. To execute an arithmetic expression in Maple, the expression must be terminated with a semicolon or colon. The # symbol is used for commenting in a Maple worksheet. Below are several examples of numerical computations with their corresponding outputs.

```
> 2 + 2;
```

4

```
> 1 + 1 / 2;
```

$\frac{3}{2}$

```

> 1 + 0.5;

1.5

> sqrt(2);                                # sqrt() takes the square root

 $\sqrt{2}$ 

> Pi;

 $\pi$ 

> evalf(Pi);

3.141592654

> 2 * 2.5:

> % + 1 / 2;

5.500000000

```

From these few examples, it is important to note the following:

- Spaces between symbols are optional, so `2 + 2;` and `2+2;` are equivalent. We include spaces between operators for readability, consistent with good programming practice.
- Maple performs exact calculations with rational numbers and approximate calculations with decimals. To Maple, the rational number $3/2$ and the floating-point approximation 1.5 are different objects. Using both decimals and rational numbers in a statement produces a decimal output.
- Maple interprets irrational numbers as exact quantities. Maple also recognizes standard mathematical constants, such as π and e , and works with them as exact quantities.
- The `evalf` command converts an exact numerical expression to a floating-point number. By default, Maple calculates the result using ten digits of accuracy, but any number of digits can be specified. The optional second argument of `evalf` controls the number of floating-point digits for that particular calculation.
- For particularly precise numerical expressions, a call to `evalhf` evaluates an expression to a numerical value using the hardware floating-point precision of the underlying system. The evaluation is done in double precision. The `evalhf` function computes only with real floating-point arguments.
- If a statement ends with a colon, instead of a semicolon, then Maple suppresses the output, although it will store the result (if assigned to a variable) internally.
- The ditto operator `%` refers to your last calculated result, even if that result is not on the line preceding the `%`.

2.2 Variables

It is convenient to assign variable names to expressions that are referred to one or more times in a Maple session. Maple's syntax for assigning a variable name is `name := expression`. Almost any expression, including numbers, equations, sets, lists, and plots, can be given a name, but it is indeed helpful to choose a name that describes the expression. The `restart` command makes Maple act (almost) as if it has just been started, and it clears the values of all Maple variables. Some guidelines for variable names are:

- Maple is case sensitive, so the names `X` and `x` denote unique variables.
- A variable name can contain alphanumeric characters and underscores, but it *cannot* start with a number.
- Once a variable is assigned to an expression, it remains that expression until changed or cleared.
- A variable name can be cleared by assigning `variable := 'variable'`; or executing the statement `unassign('variable')`;
- Maple has some predefined and reserved names, such as `Sum`, `sqrt`, and `length`, that are not available for variable assignment. Maple will not allow an expression to be assigned to a predefined variable name.
- When you close a Maple session, variable names assigned during that session will be forgotten. When a Maple worksheet is re-opened, variable names must be reactivated.
- The `restart` command at the top of the worksheet, followed by the sequence of keystrokes `Alt, e, e, w` restarts the memory of the variables and then executes the entire worksheet in order. Often one will be in the middle of a series of commands, fix an error and need to re-execute all commands in order once again.

Below are several examples of defining variable names.

```
> restart:
> EventA := 0.3:
> EventB := 0.3:
> EventC := 0.4:
> S := EventA + EventB + EventC;

S := 1.0

> Sum := EventA + EventB;
Error, attempting to assign to 'Sum' which is protected

> prob := p * (1 - p) ^ 2:
> p := 1 / 4:
> prob;
```

$$\frac{9}{64}$$

```
> unassign('p'):                                # You could also write p := 'p'
> newprob := 3 * p ^ 2 * (1 - p);

newprob := 3p2(1 - p)
```

Once a Maple expression is given a name, it can be evaluated at different values using `subs()` or `eval()`. The command `subs(p = 1 / 2, newprob)` or `eval(newprob, p = 1 / 2)` yields the value $\frac{3}{8}$. Since `newprob` is a variable, and not a function, Maple does not understand the notation `newprob(1/3)`, which the user may incorrectly try to use to determine the value of *newprob* for $p = 1/3$. If an expression is intended to actually define a function, then the function must be formally defined, and two techniques for doing so appear in Section 2.4.

Sometimes assumptions must be made on variables in order to set variable properties or relationships. A common use of the `assume` function is to assume a constant is positive, i.e., `assume(p > 0)`. Making such assumptions allow Maple routines to use this information to simplify expressions, for example, $\sqrt{p^2}$. When an assumption is made about a variable, thereafter the variable is displayed with an appended tilde \sim to indicate that it carries assumptions. The `additionally` function adds additional assumptions without removing previous assumptions. For example, we could further restrict p to be less than one with the command `additionally(p < 1)`.

2.3 Symbolic Computations

One of Maple's main strengths is its ability to manipulate symbolic expressions. Symbols can be treated in the same way that numbers were in the previous section and can do much more. Entering a constant or variable followed by a variable, e.g., `2x` or `ab`, does not imply multiplication in Maple. In fact, `ab` would be treated as a new two-letter variable instead of the product of two single-letter variables. You may *not* omit the multiplication symbol (`*`) in expressions with more than one factor. Below are a few examples of Maple's symbolic abilities using three commands, `combine`, `expand`, and `simplify`, that appear in the APPL code.

```
> exp(-t ^ 2) * exp(-s ^ 2);

e(-t2)e(-s2)

> combine(%);

e(-t2-s2)

> mgf_1 := (1 / 3) * exp(t) + (2 / 3) * exp(2 * t):
> mgf_2 := (1 / 4) + (3 / 4) * exp(-t):
```



```
> mgf_1 * mgf_2;
```

$$\left(\frac{1}{3}e^t + \frac{2}{3}e^{(2t)}\right) \left(\frac{1}{4} + \frac{3}{4}e^{-t}\right)$$

```
> expand(%);
```

$$\frac{7}{12}e^t + \frac{1}{4} + \frac{1}{6}(e^t)^2$$

```
> mgf_1 / mgf_2;
```

$$\frac{\frac{1}{3}e^t + \frac{2}{3}e^{(2t)}}{\frac{1}{4} + \frac{3}{4}e^{(-t)}}$$

```
> simplify(%);
```

$$\frac{4}{3} \frac{e^{(2t)} (1 + 2e^t)}{e^t + 3}$$

2.4 Functions

Functions are defined in Maple by using the arrow notation `->` or the `unapply()` command. The assignment operator `:=` associates a function name with a function definition. Two equivalent ways of defining a function f are

```
> f := x -> exp(-2) * 2 ^ x / x!;
```

```
> f := unapply(exp(-2) * 2 ^ x / x!, x);
```

$$f := x \rightarrow \frac{e^{(-2)}2^x}{x!}$$

This notation allows a function to be evaluated in the “usual” way, i.e., $f(0)$, when it appears in Maple expressions. Such functions are an integral part of APPL, and are used to create PDFs, CDFs, and transformations of random variables. Unassigning a function is done in the same way that a variable is unassigned to a value, `f := 'f'`. As shown in the examples below, piecewise functions may also be defined in Maple, and we certainly take advantage of this in APPL, e.g., the triangular distribution.

```
> g := unapply(exp(-lambda) * lambda ^ x / x!, lambda, x);
```

$$g := (\lambda, x) \rightarrow \frac{e^{(-\lambda)}\lambda^x}{x!}$$

```
> g(2, 0);
```

$$e^{(-2)}$$

```
> h := x -> piecewise(x <= 1, 0, x > 1, 2):
> h(x);
```

$$\begin{cases} 0 & x \leq 1 \\ 2 & x > 1 \end{cases}$$

In addition to facilitating the creation of custom functions, Maple has built-in mathematical functions, e.g., trigonometric functions such as `tan`, inverse trigonometric functions such as `arccos`, absolute value (`abs`), the exponential function (`exp`), the gamma function (`GAMMA`), and binomial coefficients (`binomial`). As illustrated earlier in the chapter, the exponential function in Maple is written as `exp(x)`. In order to compute e^3 , enter `exp(3)`, *not* `e ^ 3`. The binomial function `binomial(n, x)` determines binomial coefficients for selecting x items from n items without replacement. Examples of the functions used frequently in the APPL code are below.

```
> abs(ln(0.4) + arcsin(0.65));
0.2087062952

> binomial(4, 2);
6

> GAMMA(5);
24

> evalf(exp(2));
0.9953222650
```

Maple has packages of functions available with specialized commands to perform tasks from an extensive variety of disciplines. The commands in a package can be activated using the `with` command. For example, `with(networks)` adds the `networks` package, which supplies tools for constructing, drawing, and analyzing combinatorial networks. Likewise, `with(plots)` enables more options with plotting functions.

When a package's functions are accessed using the `with` command, occasionally a previous definition or a function, either built-in or defined by the user, will be overridden. For example, `trace` is by default defined to be the program-tracing function. However, after invoking `with(linalg)`, `trace` is redefined to mean the trace of a matrix. The `with` function prints out a warning message whenever it overrides a previous definition or function.

2.5 Data Types

This section examines some basic types of Maple objects, including lists, sets, arrays, and strings.

A Maple *list* can be created by enclosing any number of expressions (separated by commas) in brackets. Maple preserves the order and repetition in

a list. Thus, `[a,b,c]` and `[b,c,a]` are different lists. The `nops` command determines the total number of elements within a list. The membership of an element in a list can be checked with `member()`, which returns a *true* or *false* boolean statement.

```
> probs := [0.3, 0.5, 0.2]:
> nops(probs);
```

3

```
> member(0.3, probs);
```

true

Because order is preserved in a list, a particular element can be extracted from a list without searching for it. Use `L[i]` to access the i^{th} element of a list `L`. The `op` command applied to a list removes the brackets of the list, and `sort` preserves the list structure while sorting the elements into ascending order.

```
> probs := [0.3, 0.5, 0.2]:
> op(probs);
```

0.3, 0.5, 0.2

```
> sort(probs);
```

[0.2, 0.3, 0.5]

In APPL, we utilize the functionality of Maple's list structure to represent a random variable as a "list-of-sublists." A list-of-sublists is just a list, each of whose elements is itself a list. For example, we create a random variable as a list of three sublists, where the first sublist contains the functional form of the distribution, the second sublist contains its support, and the third sublist uses strings to describe the type of the random variable, i.e., continuous or discrete, and the type of function, e.g., the PDF, CDF or HF. When APPL receives a random variable for manipulation, it first checks the type of the random variable before continuing; that is, is the random variable continuous or discrete? Depending on the answer, APPL treats that random variable accordingly, for example, integrating to determine expected value versus summing. Additional information about APPL's list-of-sublists data structure for random variables is contained in Chapters 3 and 6. Elements in a list-of-sublists can be extracted using indices. For example,

```
> x := [[1, 2, 3], [4, 5, 6], [7, 8]];
```

sets `x` to a list of three elements, each of which is a list. Sublists or individual elements can be extracted as follows:

```
> x[2];
```

[4, 5, 6]

```
> x[2, 1];
```

4

```
> x[2][1];
```

4

A Maple *set* is constructed by enclosing any number of Maple objects (separated by commas) in braces. The braces identify the object as a set. Maple does not preserve order or repetition in a set. Thus the sets $\{a, b, c\}$, $\{b, c, a\}$, and $\{c, c, b, a\}$ are identical. The **member** command validates membership in a set, and to choose an item from a set, the subscript notation $[i]$ is used, where i identifies the position of the desired element in the set. Also, the **nops** function counts the number of elements in a set.

The **seq** command is used to generate sequences, which can be contained in a list or a set if desired. For example, a sequence of binomial coefficients can be obtained by using **seq** as follows:

```
> seq(binomial(4, i), i = 0 .. 4);
```

1, 4, 6, 4, 1

The Maple *array* data structure is an extension of the Maple list data structure. Each element is still associated with an index (which can be negative or zero), but an array is not restricted to one dimension.

A *string* is a Maple object created by enclosing a sequence of characters in double quotes. We use strings in APPL in the third sublist of the list-of-sublists that define a random variable to help identify its functional form, such as "Continuous", "Discrete", "PDF", "CDF", etc.

2.6 Solving Equations

Maple can find analytic solutions for a large class of algebraic equations. The **solve** command is a general-purpose equation solver. It takes a set of one or more equations and attempts to solve it exactly for the specified set of unknowns. The **fsolve** command is the numeric equivalent of **solve**; **fsolve** calculates the solution(s) to equations using a variation on Newton's method, producing approximate (floating-point) solutions. For a general equation, **fsolve** searches for a single root and often returns the first root it finds, which may not be the desired root. For a polynomial, it looks for all real roots. Fortunately, there is a way to make **fsolve** look for a specific solution by specifying an interval over which **fsolve** may search, as illustrated in the examples to follow. Some special notes about solving equations are

- If the variable(s) being solved for are not specified, Maple solves for all variables.

- If no right-hand side to an equation is given (for example, `solve(2 * x - 8)`), the right-hand side is assumed to be 0.
- The `fsolve` command often only returns one real solution.
- If there are multiple solutions to an equation, they can be extracted by setting the solution set to a variable, then using subscripts `[1]`, `[2]`, etc.

Here are several examples of solving one or more equations at a time.

```
> solve(x ^ 3 = 2 * x, x);
```

$$0, \sqrt{2}, -\sqrt{2}$$

```
> fsolve(x ^ 3 = 2 * x, x);
```

$$-1.414213562, 0., 1.414213562$$

```
> eqns := {x + 2 * y = 3, y + 1 / x = 1}:
```

```
> solns := solve(eqns, {x, y});
```

$$solns := \{y = 2, x = -1\}, \left\{x = 2, y = \frac{1}{2}\right\}$$

```
> solns[2];
```

$$\left\{x = 2, y = \frac{1}{2}\right\}$$

```
> solve(cos(x) - x = 0);
```

$$\text{RootOf}(-Z - \cos(-Z))$$

```
> fsolve(cos(x) - x = 0);
```

$$0.7390851332$$

```
> express1 := (1 / x) + (1 / x ^ 2) - (3 / x ^ 3):
```

```
> solve(express1);
```

$$-\frac{1}{2} + \frac{\sqrt{13}}{2}, -\frac{1}{2} - \frac{\sqrt{13}}{2}$$

```
> fsolve(express1);
```

$$1.30277563$$

```
> fsolve(express1, x = -3 .. 0);
```

$$-2.302775638$$

Maple occasionally returns solutions in terms of the `RootOf` function, which is a place holder for representing all the roots of an equation in one variable. The prefix `_Z` on the variable in a `RootOf` solution indicates that it

has integer values. The `fsolve` command is able to approximate these roots (at most one at a time for non-polynomial equations, and a specific root if a search interval is specified). Below is also an indication of the range of a variable using two periods in a row. The range delimiter “..” denotes endpoints of intervals in plots, integrations, and other situations.

Two other commands of interest for obtaining either the left-hand side or right-hand side of an expression are `lhs` and `rhs`, respectively. Especially useful to APPL is the ability of these commands to extract the first and last term of a sequence. This functionality allows APPL to read off the bounds on the support of a probability function, which is displayed in the second sublist for a random variable. Both uses of `lhs` and `rhs` are displayed below.

```
> eqn1 := x + y = z + 3:
> lhs(eqn1);
```

$$x + y$$

```
> rhs(eqn1);
```

$$z + 3$$

```
> support := 1 .. infinity:
> lhs(support);
```

$$1$$

```
> rhs(support);
```

$$\infty$$

2.7 Graphing

Another capability of Maple is graphing. The explicit function $y = f(x)$ and its domain must be specified for a Maple plot. Plotting the standard normal distribution on the interval $-3 \leq x \leq 3$, for example, can be done with the Maple statements shown below. We will use the function form `->` to define the PDF of the standard normal distribution. The plot is displayed in Figure 2.1.

```
> f := x -> 1 / sqrt(2 * Pi) * exp(-x ^ 2 / 2):
> plot(f(x), x = -3 .. 3);
```

In order to plot more than one function on a single graph, the functions must be contained in brackets or braces. The use of brackets means that order is important and Maple should graph them in the given order; the use of braces means that order is not important and Maple may graph them in any order. Also, it is sometimes useful to specify the range of the y -values in order to display important features of a graph. Figure 2.2 contains a plot of two normal PDFs. The functions are defined with two different variables, but used as arguments in `plot` with a third variable, as one would expect.

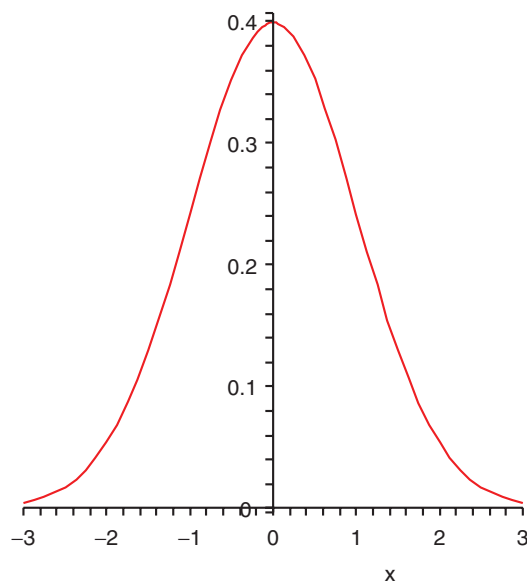


Fig. 2.1. Plot of a standard normal PDF between $x = -3$ and $x = 3$.

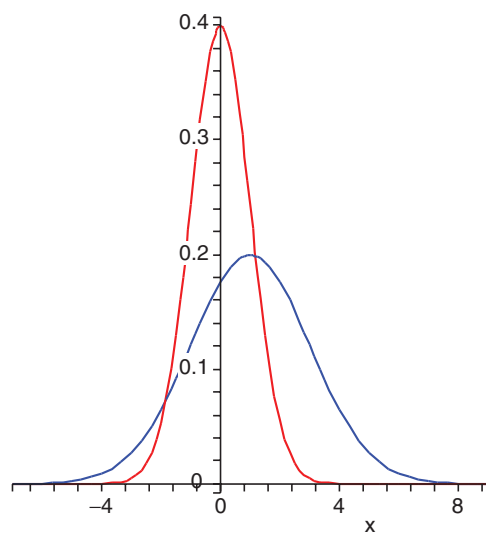


Fig. 2.2. Plot of a standard normal PDF and a normal PDF with mean $\mu = 1$ and standard deviation $\sigma = 2$ between $x = -7$ and $x = 9$.

```
> f1 := t -> 1 / sqrt(2 * Pi) * exp(-t ^ 2 / 2):
> f2 := z -> 1 / (2 * sqrt(2 * Pi)) * exp(-(z - 1) ^ 2 / 8):
> plot({f1(x), f2(x)}, x = -7 .. 9, color = [red, blue]);
```

2.8 Calculus

Maple provides many powerful tools for solving problems in calculus. This section describes Maple's ability to determine limits, symbolically compute derivatives and definite integrals, and determine summations. The need for calculus operations in APPL is great because many probability theorems rely on calculus to create new distribution functions. A few comments about determining limits, derivatives, and definite integrals in Maple follow:

- Maple will compute both two-sided limits, e.g., $\lim_{x \rightarrow 5} \left(\frac{x+2}{x-1} \right)$, and one-sided limits, e.g., $\lim_{x \uparrow 1} \left(\frac{x+2}{x-1} \right)$, with the `limit` procedure.
- Maple can determine limits at ∞ , such as the limit of an exponential CDF at infinity.

Examples of the `limit` command are

```
> limit((3 ^ (x + h) - 3 ^ x) / h, h = 0);
3^x ln(3)

> assume(lambda > 0):
> limit(1 - exp(-lambda * x), x = infinity);
```

1

The `diff` procedure computes the derivative of an expression. Maple differentiates with respect to the variable placed in the second argument of the `diff()` procedure. Clearly, `diff` is used often in APPL for changing from one functional form of a continuous random variable's distribution to another, e.g., deriving a PDF from a CDF. One such example of the `diff()` command follows; however it is important to note that APPL will do such conversions in a more automated manner. Also note the use of the `unapply` command to turn the final expression into a function.

```
> F := x -> 1 - exp(-2 * x);
F := x → 1 - e(-2x)

> f := unapply(diff(F(x), x), x);
f := x → 2e(-2x)
```

The capabilities of Maple include a robust set of integration operations. Symbolic integrals (often used in APPL) can be calculated as in the following example:

```
> f := y -> 2 * exp(-2 * y):
> F := unapply(int(f(y), y = 0 .. x), x);
```


$$F := x \rightarrow 1 - e^{(-2x)}$$

Maple's ability to integrate also includes the numerical evaluation of integrals for which no closed-form solution is available, such as the case with the standard normal random variable:

```
> g := 1 / sqrt(2 * Pi) * exp(-x ^ 2 / 2):
> int(g, x = -infinity .. 1.96);

0.9750021049
```

However, if we integrate the standard normal PDF over exact values, rather than floating point values, Maple responds with **erf**. For example, integrating a standard normal PDF from $x = -1$ to 1 yields the following:

```
> int(g, x = -1 .. 1);


$$\operatorname{erf}\left(\frac{\sqrt{2}}{2}\right)$$

```

The function **erf**(x) is the error function and it is defined for real and complex x by $\operatorname{erf}(x) = \frac{2}{\pi} \int_0^x e^{-t^2} dt$. Maple can numerically evaluate an **erf**(x) output with the **evalf** command.

```
> evalf(int(g, x = -1 .. 1));

0.6826894920
```

Maple also can compute finite sums and infinite series, which are necessary capabilities for working with discrete random variables. The next few examples demonstrate the use of **sum**. The first example verifies that a geometric distribution has a valid PDF.

```
> geometric := (1 / 4) * (3 / 4) ^ (x - 1);


$$geometric := \frac{\left(\frac{3}{4}\right)^{(x-1)}}{4}$$


> sum(geometric, x = 1 .. infinity);

1
```

The second example determines the expected value of a binomial distribution with parameters n and p .

```
> assume(p > 0):
> additionally(p < 1):
> pdf := binomial(n, x) * p ^ x * (1 - p) ^ (n - x):
> simplify(sum(pdf * x, x = 0 .. n));

np~
```

2.9 Loops and Conditions

Like all programming languages, loops in Maple allow the user to execute a sequence of statements repeatedly, either for a prescribed number of times or until a condition is satisfied (or not satisfied). This is a useful tool in APPL when working with random variables. Loops are used in APPL to do many things: cycle through segments of PDFs, verify inverse transformations, and error-checking, to name a few. For example, if a discrete random variable's PDF consists of a finite number of numeric probabilities in the first sublist in a list-of-sublists, we require these probabilities to be non-negative. Using a **for** loop allows APPL to cycle through the items in this sublist (**from** item one **to** **nops(sublist)**) until the list is complete or a negative value is identified. Similarly, if numeric probabilities in the first sublist are supposed to be equally-likely probabilities, then APPL can repeatedly check each probability value while the probability values remain the same. The expression in the **while** clause is a Boolean expression which must evaluate to *true* or *false* upon each cycle.

Conditional statements in APPL have a variety of purposes including error checking, branching in logic, and conditional plotting. If an APPL procedure requires input arguments from the user, then an **if** statement can be used to verify that the input is valid. If a procedure does not receive the correct type of arguments, e.g., lists, an error message is sent back to the APPL user. Also, in the third sublist is the “form” of the random variable. The first element of the third sublist tells APPL whether the random variable is continuous or discrete. If the random variable is continuous, then continuous mathematics (e.g., integration) is used to verify the validity of its PDF or compute its mean, for example. If discrete random variables are indicated by the field, then discrete mathematical techniques (e.g., summation) are used.

Boolean expressions in the conditional statement are most often formed by using **<**, **<=**, **>**, **>=**, **=**, **<>**, **or**, **and**, or **not**. For example, if the “form” of the random variable is **not** one of the approved formats (e.g., PDF, CDF, SF, HF, CHF, IDF), then an error message is reported to the user.

A few examples of small loops and conditionals are presented here to amplify the above explanations:

```
> probs := [0.1, 0.05, 0.2, 0.25, 0.15, 0.10, 0.15]:
> totprob := 0:
> for i from 1 to nops(probs) do
>   totprob := totprob + probs[i]:
> end do;           # 'od' can also be used to end a 'do' loop

totprob := 0.1
totprob := 0.15
totprob := 0.35
totprob := 0.60
```

```

totprob := 0.75
totprob := 0.85
totprob := 1.00

```

```

> die := rand(1 .. 6):           # random integer from 1 to 6
> Roll1 := die():
> Roll2 := die():
> if (Roll1 >= Roll2) then
>   print("Roll 1 is greater than or equal to Roll 2");
> else
>   print("Roll 1 is less than Roll 2");
> end if; # 'fi' can also be used to end an 'if' conditional

```

To conclude this section, a **while** loop is created in which the user is encouraged to experiment with various values for the probability *prob*.

```

> i := 1:
> totprob := 0:
> prob := 1 / 2:
> while (totprob <= 1) and (i <= 1000) do
>   totprob := prob + totprob:
>   prob := prob ^ 2 + 0.001:
>   i := i + 1:
>   print(i):
>   print(totprob):
> end do:

```

When a **break** statement is used in a procedure, a break is executed and the result is to exit from the innermost repetition (**for/while/do**) statement within which it occurs.

2.10 Procedures

The capability to create new procedures in Maple is a key element to enabling APPL programming. In fact, APPL is a text file of new Maple procedures that is imported into a worksheet session with the **read** command. When a file is read into Maple, each line in the file is treated as if it had been typed in order. The APPL procedures are available during the Maple session, but the commands are not displayed on the Maple worksheet. When the user reads in the APPL commands, it's as if they used the **with** command to add extra capability. The syntax for reading APPL into Maple from the ASCII source code file named `APPL.txt` is `read('APPL.txt')`.

The syntax for creating a procedure in Maple is

```

proc(<argseq>) [local <nseq>; global <nseq>;]

```

⟨statement sequence⟩

end;

where [] indicates optional parameters. Then

- *⟨argseq⟩* is a sequence of variable names, separated by commas. These variables are the arguments to the procedure.
- *⟨nseq⟩* is a sequence of variable names, separated by commas.
- *local ⟨nseq⟩* is a list of variables local to the procedure. Any assignments of these variables will only have a scope of the procedure. The values of these variables will be unassigned when the procedure starts, regardless of the value of the variable outside the procedure.
- *global ⟨nseq⟩* is a list of global variables used by the procedure. Any assignments made in the procedure will be global in scope and the initial values of the variables will be as they are in the existing Maple session.
- *⟨statement sequence⟩* is the body of the procedure and may consist of any valid sequence of Maple statements.

The **proc** command is usually used in conjunction with the **RETURN** keyword. The syntax is

RETURN (*expression*);

This command will terminate the execution of the procedure and return the value of *expression*.

Since APPL's random variables often have quite a few arguments, it's helpful that Maple provides an easy way to do data typing in procedures. The arguments passed to a procedure may be given a specific data type. If any of these variables is then assigned an incorrect type, Maple generates an error message. The syntax for declaring a variable to be of a given data type is the declaration is

variable name :: data type

where Maple accepts a wide range of data types. The types most frequently encountered in APPL are **list** and **listlist** (because of the list-of-sublists structure of APPL random variables), **constant**, **posint**, **int**, and **array**. Another type used frequently in APPL is **symbol**. Infinity (∞) is of type **symbol**, and this is important information for APPL procedures to know when evaluating random variables over supports that are not finite. The type of an expression can be determined in Maple with the command **type**. The syntax is

type (*expression, type*)

which returns a boolean value, either *true* or *false*.

Another useful error-checking command that can be used with procedures is **nargs**, which is the number of arguments passed to a procedure. We use

if statements in APPL to determine if optional arguments are passed to a procedure and how the procedure should proceed given this number of arguments. For example, the `OrderStat` procedure in APPL will perform differently based on whether or not it is given an optional fourth argument that specifies whether sampling is done without replacement.

To summarize the last few sections on programming in Maple, the following excerpt from the APPL source code is noteworthy. The procedure is called `ReduceList` and it is a small sub-procedure in APPL that looks for redundant support entries in a random variable list-of-sublist's second sublist. One sees how the `proc` command begins the procedure as well as some argument checking, local variable declarations, a `for` loop and some conditional branching. The `RETURN` and `end` commands end the procedure.

```
#
# ReduceList is a procedure that eliminates floating point
# redundancies (e.g., 3 vs. 3.0) from a sorted Maple list.
#
ReduceList := proc(LST :: list)
local i, size, delt, deltamin, ListIn:
deltamin := 0.0000001:
ListIn := LST:
size := nops(ListIn):
for i from (size - 1) by -1 to 1 do
    if (ListIn[i] <> -infinity and ListIn[i + 1] <> infinity) then
        delt := evalf(ListIn[i + 1]) - evalf(ListIn[i]):
        if (delt < deltamin) then
            if (whattype(ListIn[i]) <> float) then
                ListIn := subsop((i + 1) = NULL, ListIn):
            else
                ListIn := subsop(i = NULL, ListIn):
            fi:
        fi:
    fi:
od:
RETURN(ListIn):
end:
```

This concludes our brief introduction to the Maple computer algebra system. Additional help on a specific Maple topic can be accessed from the Maple worksheet by entering `?topic` or `help(topic)` at the prompt. Also, help can be obtained directly by going to the “Help” menu in Maple, scrolling down to “Topic Search,” and entering the topic of interest in the “Topic” line. We now turn to the main topic of the monograph, defining random variables in APPL and the associated algorithms to manipulate them. We begin with continuous random variables.

Data Structures and Simple Algorithms

This chapter and the two that follow it are concerned with continuous random variables. We have chosen to present continuous random variables first because they are defined with a somewhat simpler data structure than that for discrete random variables. The development described here gives a probabilist the ability to automate the instantiation and processing of continuous random variables—key elements of computational probability.

3.1 Data Structures

There are dozens of ways that one could represent the distribution of a random variable in Maple. Whatever choice is made, the syntax associated with the data structure should be succinct, intuitive, and easily manipulated. This implies there should be a single data structure that allows a user to define the

- probability density function (PDF) $f_X(x)$
- cumulative distribution function (CDF) $F_X(x) = \int_{-\infty}^x f_X(s) ds$,
- survivor function (SF) $S_X(x) = 1 - F_X(x)$
- hazard function (HF) $h_X(x) = f_X(x)/S_X(x)$
- cumulative hazard function (CHF) $H_X(x) = \int_{-\infty}^x h_X(s) ds$, and
- inverse distribution function (IDF) $F_X^{-1}(u)$

for a continuous random variable X . It should also provide a seamless transition from one representation to another. The common data structure that we used in the development of the APPL software package is referred to as the “list-of-sublists.” Specifically, any functional representation of a random variable is presented in a list that contains three sublists, each with a specific purpose. The first sublist contains the ordered functions that define the functional forms of the distribution. The PDF representation of the triangular distribution, for example, would have the two linear functions that comprise

the two segments of its PDF for its first sublist. The second sublist is an ordered list of real numbers that delineate the end points of the intervals for the functions in the first sublist. The end point of each interval is assumed to be the starting point of the succeeding interval. The third sublist has two elements. The first element of the third sublist is either the string "Continuous" for continuous distributions or "Discrete" for discrete distributions. The second element of the third sublist shows which of the six functional forms listed above is used in the first sublist. The string "PDF", for example, indicates that the first sublist denotes a PDF.

Example 3.1. The following seven random variables give a variety of ways continuous random variables can be defined in APPL.

- The following APPL statement assigns a list-of-sublists that represents the PDF of a $U(0, 1)$ random variable to the list X:

```
> X := [[x -> 1], [0, 1], ["Continuous", "PDF"]];
```

- The triangular distribution has a PDF with two segments to its distribution. The following statement defines a triangular(0, 1, 2) random variable named X with a minimum of 0, mode of 1, and maximum of 2, as a list-of-sublists:

```
> X := [[x -> x, x -> 2 - x], [0, 1, 2],
        ["Continuous", "PDF"]];
```

- An exponential random variable X with a mean of 1/3 can be defined in terms of its hazard function with the APPL statement

```
> X := [[x -> 3], [0, infinity], ["Continuous", "HF"]];
```

- The three random variables defined above all have numeric parameters. APPL is also able to accommodate symbolic parameters. A $N(\theta, 1)$ random variable X can be defined with the statement

```
> X := [[x -> exp(-(x - theta) ^ 2 / 2) / sqrt(2 * Pi)],
        [-infinity, infinity], ["Continuous", "PDF"]];
```

- The parameter space can be specified by using the Maple `assume` function. Consider the random variable T with HF

$$h_T(t) = \begin{cases} \lambda & 0 < t < 1 \\ \lambda t & t \geq 1 \end{cases}$$

for $\lambda > 0$. The random variable T can be defined by the statements

```
> assume(lambda > 0);
> T := [[t -> lambda, t -> lambda * t],
        [0, 1, infinity], ["Continuous", "HF"]];
```

- The syntax allows the endpoints of intervals associated with the support of a random variable to be specified symbolically. A $U(a, b)$ random variable X is defined by

```
> assume(a < b);
> X := [[x -> 1 / (b - a)], [a, b],
        ["Continuous", "PDF"]];
```

- There is no error checking to ensure that the distribution associated with a random variable is legitimately represented by a particular list-of-sublists. This means that the statement shown below will create a list-of-sublists that is not a legitimate PDF.

```
> X := [[x -> 17], [0, 49], ["Continuous", "PDF"]];
```

An APPL procedure named `VerifyPDF` will be introduced that can be used to verify the PDF properties for a random variable.

After repeatedly typing the list-of-sublists for common continuous distributions, we soon realized that the list-of-sublists format could be automated for certain common (e.g., normal, exponential) distributions. We appended the letters RV onto these procedure names so as to avoid any conflicts with existing Maple procedures. The following continuous distributions have a procedure that creates the appropriate list-of-sublists in PDF form for use in APPL: *ArcSin*, *ArcTan*, *Beta*, *Cauchy*, *Chi*, *ChiSquare*, *Erlang*, *Error*, *Exponential*, *ExponentialPower*, *ExtremeValue*, *F*, *Gamma*, *GeneralizedPareto*, *Gompertz*, *HyperbolicSecant*, *HyperExponential*, *HypoExponential*, *IDB*, *InverseGaussian*, *InvertedGamma*, *KS*, *LaPlace*, *LogGamma*, *Logistic*, *LogLogistic*, *LogNormal*, *Lomax*, *Makeham*, *Muth*, *Normal*, *Pareto*, *Rayleigh*, *StandardCauchy*, *StandardNormal*, *StandardTriangular*, *StandardUniform*, *T*, *Triangular*, *Uniform*, and *Weibull*. This list of distributions, along with their parameters, is displayed every time the APPL code is read into Maple. A chi square random variable with six degrees of freedom, denoted by X , for example, can be defined by the APPL statement

```
> X := ChiSquareRV(6);
```

This is much more compact than the standard list-of-sublists approach:

```
> X := [[x -> x ^ 2 * exp(-x / 2) / 16], [0, infinity],
        ["Continuous", "PDF"]];
```

The `ChiSquareRV` procedure (or function, to use the more standard parlance) relieves the user of typing the list-of-sublists every time a chi square random variable is required. The APPL source code that defines a chi square random variable with n degrees of freedom, complete with error trapping, is given by:


```

> ChiSquareRV := proc(n :: {posint, symbol})
> local ListOfLists, x:
> global n1:
> n1 := n:
> if (nargs <> 1) then
>   print('ERROR: This procedure requires 1 argument'):
>   RETURN():
> fi:
> if type(n1, infinity) then
>   print('ERROR: The shape parameter must be finite'):
>   RETURN():
> fi:
> if type(n1, symbol) then
>   assume(n1, posint):
>   about(n1):
> fi:
> ListOfLists := [[unapply(x ^ (n1 / 2 - 1) * exp(-x / 2) /
    (2 ^ (n1 / 2) * GAMMA(n1 / 2)), x)],
    [0, infinity], ["Continuous", "PDF"]]:
> RETURN(ListOfLists):
> end:

```

We use Maple's `::` in the procedure call just after the argument for Maple error checking to ensure that the argument is either a positive integer or a symbol. If the argument is a symbol, Maple appends a tilde (\sim) to the symbol as a reminder to the user that an assumption (in this case that it is a positive integer) will be placed on the symbol. The introduction of global variables for parameters allows any assumptions about the parameters to be passed from APPL code to the worksheet one is using. For example, the APPL statement

```
> X := ChiSquareRV(6);
```

returns a list-of-sublists with 6 substituted for `n1` in the PDF. On the other hand, the APPL statement

```
> X := ChiSquareRV(m);
```

returns a list of lists with $m \sim$ substituted for `n1` in the PDF. This allows for certain symbolic operations to be done subsequently on `X`, such as verifying (symbolically) that the area under the PDF is one. The first two `if` statements in `ChiSquareRV` check that there is just one argument passed to the procedure and that the number of degrees of freedom is not passed as `infinity`. The third `if` statement places an assumption on the parameter if it is passed as a symbol and the Maple `about` procedure prints information about the assumption to the screen. Finally, the Maple `unapply` procedure transforms the expression for the PDF into the `x ->` format consistent with the earlier examples, and the PDF is returned as a list-of-sublists. The PDF for the chi

square distribution contains the gamma function, which accounts for the use of the Maple **GAMMA** procedure.

This concludes our description of the data structure that is used to represent a continuous random variable. We now briefly outline some simple algorithms that can be applied to this data structure.

3.2 Simple Algorithms

The algorithms described in this section conduct the following elementary operations on data structures that represent continuous random variables:

- convert any functional representation (PDF, CDF, SF, HF, CHF, and IDF) of a random variable into another functional representation using the common data structure,
- plot any of the six functional forms of any fully-specified distribution, e.g., the CDF,
- determine common summary characteristics of random variables, such as the mean and variance, and
- calculate the PDF of the r^{th} order statistic from a sample of n iid random variables.

Changing Functional Forms of a Random Variable. The functional form of a distribution can be converted to and from any of its six well-known forms, the PDF, CDF, SF, HF, CHF, and IDF. The routines return functional forms when they are called with a single argument and scalars when a second optional argument is included. The names of the conversion procedures are PDF, CDF, SF, HF, CHF, and IDF, and can be applied to both discrete and continuous random variables. The first example finds a CDF from a PDF for a continuous random variable.

Example 3.2. Let $X \sim U(0, 4)$. Find the CDF of X .

The APPL code required to define the distribution of X (which will be in PDF form) and determine the CDF is

```
> X := UniformRV(0, 4);
> CDF(X);
```

which returns the list-of-sublists

$$[[x \rightarrow x/4], [0, 4], ["Continuous", "CDF"]]$$

as expected.

Since APPL uses Maple's symbolic processing capability, it will typically return the appropriate function without error whenever the function can be expressed analytically. As the next example shows, however, there are many elementary examples where Maple is not able to integrate functions analytically.

Example 3.3. Let $Z \sim N(0, 1)$. Find the CDF of Z and $P(Z < 1.96)$.

Analogous to the last example, the APPL statements required to define the distribution of Z and determine its CDF are

```
> Z := NormalRV(0, 1);
> CDF(Z);
```

Since the CDF for a standard normal random variable is not closed form, APPL returns

$$\left[\left[x \rightarrow \frac{1}{2} \operatorname{erf} \left(\frac{1}{2} x \sqrt{2} \right) + \frac{1}{2} \right], [-\infty, \infty], [\text{"Continuous"}, \text{"CDF"}] \right]$$

where **erf** is the Maple error function defined by $\operatorname{erf}(x) = \frac{2}{\pi} \int_0^x e^{-t^2} dt$. To find $P(Z < 1.96)$ requires an additional APPL statement which uses the second optional argument in the call to **CDF**:

```
> prob := evalf(CDF(Z, 1.96));
```

which sets the Maple variable **prob** to 0.9750021048. The Maple **evalf** function evaluates the value of **CDF(Z, 1.96)** as a floating-point (decimal) value. A second optional argument can be included in the **evalf** function to control the number of digits displayed.

The previous examples have considered numeric parameters. The six conversion procedures work on distributions with symbolic parameters as well.

Example 3.4. Find the survivor function (SF) of a Weibull(λ, κ) random variable and find the 99th percentile of a Weibull random variable with $\lambda = 2$ and $\kappa = 3$.

For the case of symbolic parameters, the following statements define a Weibull random variable in PDF form and convert it to a Weibull random variable in SF form:

```
> X := WeibullRV(lambda, kappa);
> X := SF(X);
```

which sets **X** to the list-of-sublists format shown below:

$$\left[\left[x \rightarrow e^{(-x^{\kappa} \lambda^{-\kappa})} \right], [0, \infty], [\text{"Continuous"}, \text{"SF"}] \right]$$

As indicated earlier, the tildes after the parameters indicate that assumptions have been made concerning the parameters (i.e., $\lambda > 0$ and $\kappa > 0$) in the **WeibullRV** procedure. Finding a percentile of a distribution requires the **IDF** procedure. The 0.99 percentile of a Weibull(2, 3) distribution can be found with the APPL statement

```
> IDF(WeibullRV(2, 3), 0.99);
```

which returns 0.8318631745. If 0.99 is replaced by a random number, a Weibull(2, 3) random variate is generated. Maple automatically senses that 0.99 is floating point. The nearly-identical APPL statement

```
> IDF(WeibullRV(2, 3), 99 / 100);
```

returns the expression $2^{-2/3}(\ln(10))^{1/3}$, rather than the floating point approximation, for the 99th percentile.

The conversion procedures handle distributions defined in a piecewise fashion in the same way as those defined over a single interval.

Example 3.5. Find the SF for the random variable T with hazard function

$$h_T(t) = \begin{cases} \lambda & 0 < t < 1 \\ \lambda t & t \geq 1 \end{cases}$$

for $\lambda > 0$.

The following APPL statements are used to find the survivor function:

```
> assume(lambda > 0);
> T := [[t -> lambda, t -> lambda * t],
        [0, 1, infinity], ["Continuous", "HF"]];
> SF(T);
```

which returns

$$S_T(t) = \begin{cases} e^{-\lambda t} & 0 < t < 1 \\ e^{-\lambda(t^2+1)/2} & t \geq 1 \end{cases}$$

as expected.

The conversion procedures can be nested; for example, the APPL statements

```
> X := ExponentialRV(1);
> X := PDF(CDF(HF(SF(CHF(IDF(X))))));
```

do absolutely nothing to the list-of-sublists representation for X because all transformations can be performed analytically.

Most of the code for the six conversion routines consists of error checking, for example, to assure that

- there are one or two arguments passed,
- the distribution of interest is in the list-of-sublists format and there are three sublists, and
- the list-of-sublists corresponds to one of the six forms.

Once the appropriate `if` statements have concluded that we are converting, for example, from a CDF to a PDF, which is done by simply differentiating, the real “work” in the PDF procedure consists of just a single loop shown in the code segment

```

> for i from 1 to nsegments do
>   ffx := unapply(diff(X[1][i](x), x), x):
>   fX := [op(fX), op([unapply(simplify(ffx(x)), x)])]:
> od:

```

The Maple variable `nsegments` is a count of the number of piecewise segments associated with a PDF (e.g., `nsegments` is 2 for a triangular distribution). The first statement in the loop differentiates the i th piecewise function in the CDF, then uses the Maple `unapply` procedure to place it in the `x ->` format for the list-of-sublists. The second statement in the loop simplifies the segment of the PDF using the Maple `simplify` procedure, then concatenates the function into the list named `fX`, which is eventually returned as the first sublist in a list-of-sublists containing the PDF.

Plotting Distributions. APPL contains the `PlotDist` procedure that plots any distribution represented in the list-of-sublists format.

Example 3.6. The statements used to plot the PDF for the triangular(1, 2, 4) distribution are:

```

> X := TriangularRV(1, 2, 4);
> PlotDist(X);

```

The Maple plot drawn by the APPL `PlotDist` procedure is shown in Figure 3.1.

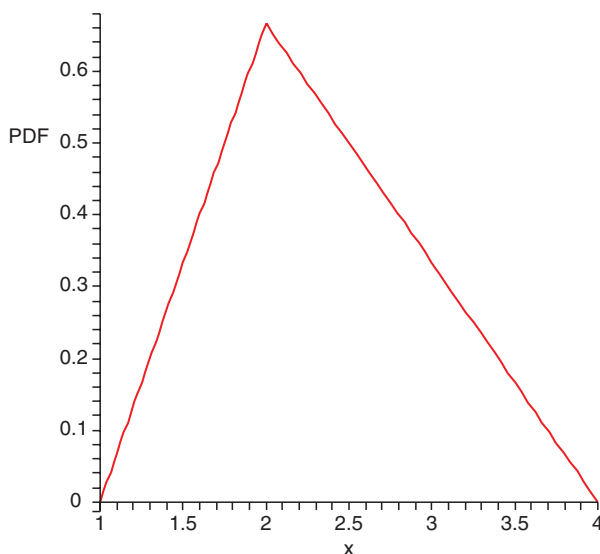


Fig. 3.1. Triangular PDF plot.

APPL is not able to plot “generic” distributions with symbolic parameters. Attempting such a plot will produce an “empty plot” error message.

Calculating Expected Values. Expected values can be calculated with the `ExpectedValue` procedure. The procedures `Mean`, `Variance`, `Skewness`, `Kurtosis`, `MGF`, and `CF` all produce these special cases of expectation by invoking the `ExpectedValue` procedure.

Example 3.7. Find the expected value, kurtosis, and moment generating function of a normal random variable X .

The second argument in the `NormalRV` procedure is σ , unlike the customary notation for a normal random variable. The APPL code to compute the mean, kurtosis, and moment generating function is

```
> X := NormalRV(mu, sigma);
> Mean(X);
> Kurtosis(X);
> MGF(X);
```

which returns μ for the mean, 3 for the kurtosis, and

$$M_X(t) = e^{\mu t + \sigma^2 t^2 / 2} \quad -\infty < t < \infty$$

for the moment generating function.

The `ExpectedValue` procedure can be used to find the expected value of a function of a random variable.

Example 3.8. Find the expected value of the square root of Y , where $Y \sim \text{exponential}(\lambda)$.

In this case, the APPL procedure `ExpectedValue` is used with a second argument that defines the function of the random variable of interest. The APPL statements

```
> Y := ExponentialRV(lambda);
> ExpectedValue(Y, y -> sqrt(y));
```

return $\sqrt{\pi}/(2\sqrt{\lambda})$.

The `ExpectedValue` procedure calculates the expected value of a function of a random variable by calculating the appropriate integrals over each segment of the support of the distribution, then summing the results.

Example 3.9. To find the mean of a triangular(a, b, c) random variable X , the APPL statements

```
> Y := TriangularRV(a, b, c);
> Mean(Y);
```

return

$$\frac{a + b + c}{3},$$

which is computed as a sum of the integrals over the two segments of the distribution as

$$E[X] = \int_a^b x \frac{2(x-a)}{(c-a)(b-a)} dx + \int_b^c x \frac{2(c-x)}{(c-a)(c-b)} dx.$$

As Example 3.9 illustrates, the essence of the computation associated with finding the expected value of any continuous random variable is to initialize the desired expected value to zero and run a loop over the number of segments, incrementing (or decrementing) the expected value by the appropriate integral.

Computing the Distribution of Order Statistics. The `OrderStat` procedure returns the PDF of the r^{th} of n order statistics drawn from a population having the same distribution as the random variable X . For a random variable X having support on $a < x < b$ with PDF $f(x)$, CDF $F(x)$, and SF $S(x)$, the PDF of the r^{th} of n order statistics drawn from this distribution is given by Hogg et al. [37, page 240] as

$$f_r(x_{(r)}) = \frac{n!}{(r-1)!(n-r)!} F(x_{(r)})^{r-1} f(x_{(r)}) S(x_{(r)})^{n-r} \quad a < x_{(r)} < b.$$

This well-known result has been implemented in an APPL procedure named `OrderStat(X, n, r)`, where X is a list-of-sublists containing the population distribution, n is the sample size, and r is the order statistic of interest.

Example 3.10. Find the distribution of the minimum of eight iid exponential(1) random variables.

The distribution of the minimum is found with the following APPL statements:

```
> X := ExponentialRV(1);
> Y := OrderStat(X, 8, 1);
```

Not surprisingly, these statements place the exponential distribution with a rate of eight into the variable Y . The list-of-sublists for Y is

$$[[x \rightarrow 8e^{-8x}], [0, \infty], ["Continuous", "PDF"]]$$

For many distributions, the `OrderStat` procedure can accommodate distributions with symbolic parameters.

Example 3.11. Find the distribution of the median of a sample of size $n = 3$ drawn from an exponential(λ) population.

In this example, an exponential distribution with a symbolic argument is entered as an argument for `OrderStat`.

```
> X := ExponentialRV(lambda);
> OrderStat(X, 3, 2);
```

The result is the symbolic PDF of the sample median:

$$[[x \rightarrow 6\lambda^2(1 - e^{-\lambda x})e^{-2\lambda x}], [0, \infty], ["Continuous", "PDF"]]$$

All of the standard programming constructs available in Maple, such as loops and conditions, can be used alongside APPL statements, as illustrated in the next example.

Example 3.12. To plot a progression of the five PDFs of the order statistics for a sample of $n = 5$ observations drawn from an exponential(1) population, use the APPL code

```
> X := ExponentialRV(1);
> n := 5;
> for i from 1 to n do
>   PlotDist(OrderStat(X, n, i), 0, 6);
> od;
```

The result is five PDFs plotted sequentially. The optional second and third arguments to `PlotDist` define the lower and upper plotting limit. This sequence could be of use to an instructor explaining the progressive nature of order statistics to first-year probability students. The last of the five plots is given in Figure 3.2.

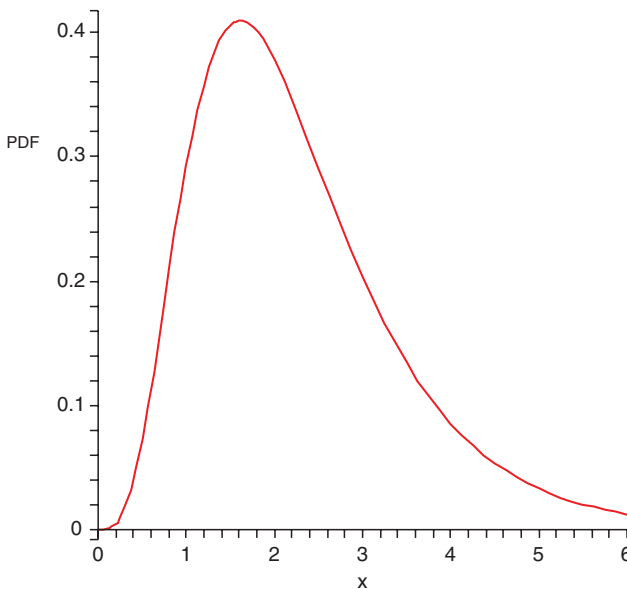


Fig. 3.2. PDF of the largest of five unit exponential random variables.

This chapter has presented data structures for representing continuous random variables and illustrated some of the simpler procedures comprising the core of APPL. The procedures shown in the APPL tree diagram in Figure 3.3 provide a roadmap for the rest of the monograph. The next two chapters describe two of APPL's nontrivial procedures that can be applied to continuous random variables: the **Transform** procedure, which finds the distribution of a transformation of a given random variable, and the **Product** procedure, which finds the distribution of the product of two given random variables.

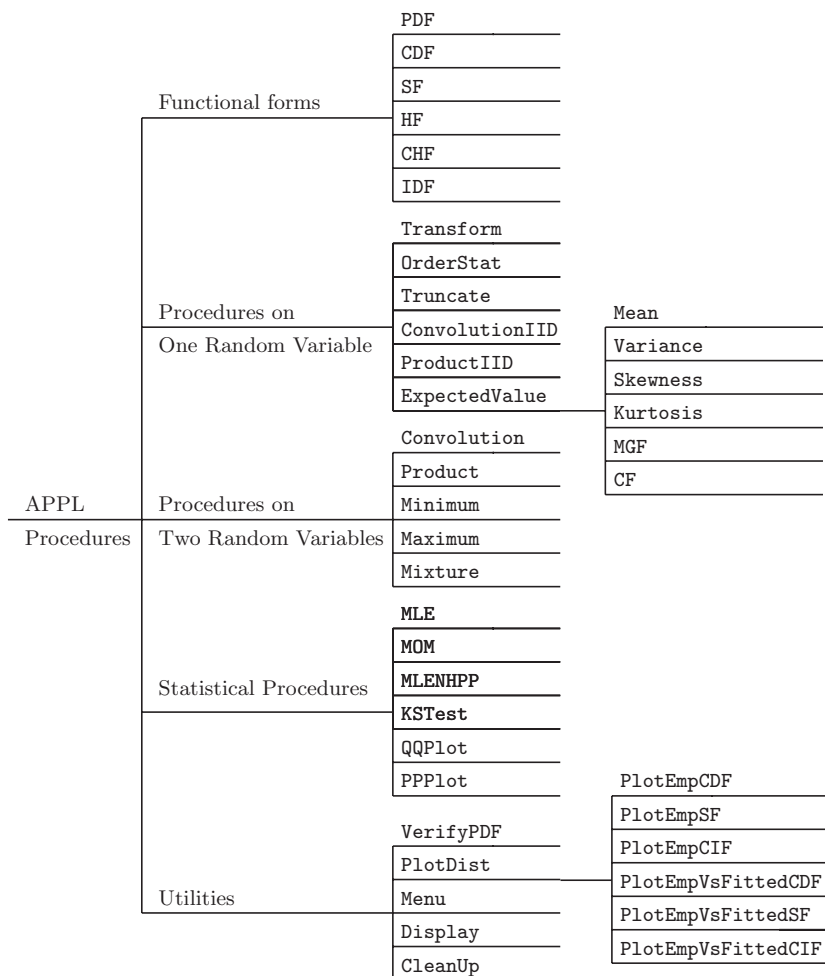


Fig. 3.3. The APPL tree.

Transformations of Random Variables

This chapter presents a generalized version of the univariate change-of-variable technique for transforming continuous random variables. Extending a theorem from Casella and Berger [12] for many-to-1 transformations, we consider more general univariate transformations. Specifically, the transformation can range from 1-to-1 to many-to-1 on various subsets of the support of the random variable of interest. We also present an implementation of the theorem in APPL and present four examples.

In its simplest application, the change-of-variable technique is used to determine the distribution of a continuous random variable Y given the distribution of a continuous random variable X and a 1-to-1 transformation from the support of X to the support of Y . As the conditions on the transformation $Y = g(X)$ become more general, the technique requires more detail in its description and is less likely to appear in introductory probability and statistics texts. Casella and Berger [12, page 53] discuss transforming random variables using the change-of-variable technique when the entire transformation is many-to-1, except for a finite number of points, i.e., the cardinality of the set $g^{-1}(y)$ is the same for almost all y in the support of Y . Hogg, et al. [37, pages 128–129] extend this many-to-1 technique to n -dimensional random variables. We are concerned with a more general univariate case in which the transformations are “piecewise many-to-1,” where “many” may vary based on the subinterval of the support of Y under consideration. We state and prove a theorem for this case and present APPL code to implement the result. Although our theorem is a straightforward generalization of Casella and Berger’s theorem, there are a number of details that have to be addressed in order to produce an algorithm for finding the PDF of Y .

Consider the following simple example. Let

$$f_X(x) = \frac{1}{3} \quad -1 < x < 2,$$

be the PDF for the random variable X . Consider the transformation $Y = g(X) = X^2$. This transformation is a 2-to-1 transformation on the interval

$X \in (-1, 1)$ (except at $X = 0$) and it is 1-to-1 on the interval $X \in [1, 2)$; see Figure 4.1 in Section 4.3. Some introductory probability texts use this as an example (e.g., Larsen and Marx [46, page 174]), but fail to state a general theorem that treats such piecewise many-to-1 transformations. The difficulty lies in identifying an appropriate partition of the support of X and a corresponding partition for the support of Y , and then determining which of these subsets of the support of X correspond to each of the subsets of the support of Y . A further complication is encountered when the transformation itself is either discontinuous or non-differentiable at certain points. For example, consider the random variable X , where $f_X(x) = (x + 1)/18$ for $-1 < x < 5$, and the transformation (see Figure 4.3 in Section 4.3)

$$Y = g(X) = \begin{cases} X^2 & -1 < X < \frac{3}{2} \\ X & \frac{3}{2} < X < 5. \end{cases}$$

In this example, the transformation is discontinuous as well as “piecewise many-to-1.” Our theorem and resulting APPL implementation will determine the PDF of Y for such transformations.

In Section 4.1, we will present our transformation theorem, modeled after Casella and Berger’s Theorem 2.1.8 [12, page 53]. Our theorem has been presented more generally in earlier textbooks. Barr and Zehna [7, page 225] consider multivariate many-to-one transformations. Rohatgi [73, pages 73–74] and Port [71, page 462] consider the piecewise many-to-one case in the univariate and multivariate settings, respectively. Our theorem is strictly univariate, but permits implementation in APPL. It determines the distribution of $Y = g(X)$ for any univariate random variable X of the continuous type with few restrictions on the transformation $g(X)$. Our theorem will be stated in a somewhat elaborate form for the purpose of facilitating its implementation.

4.1 Theorem

Before we present the theorem, we overview the rationale for the notation. We assume that the support of X , denoted by \mathcal{X} , consists of a finite union of open intervals. The points $x_1 < x_2 < \cdots < x_{n+1}$ generate n consecutive subintervals and are determined as follows. The first subinterval of \mathcal{X} begins with x_1 and the last subinterval of \mathcal{X} ends with x_{n+1} . The remaining x_i ’s correspond to other endpoints of the intervals of \mathcal{X} or to the locations in \mathcal{X} where the function g is discontinuous or non-differentiable, or g' is zero. Let $g_i(x)$ denote the restriction of $g(x)$ to (x_i, x_{i+1}) ; by design, g_i is monotone and therefore invertible. Let $X^* = \{x_1, x_2, \dots, x_{n+1}\}$; note that since X is continuous, $P(X \in X^*) = 0$.

We introduce Y^* , a set of points on the y -axis, to partition the support of Y into subintervals. The range of each g_i , denoted by (m_i, M_i) , either contains or is disjoint from each Y subinterval. The set Y^* is designed so that the

final PDF of Y may be specified by designating its value on each Y subinterval. The set Y^* is defined using one-sided limits because the transformations g_1, g_2, \dots, g_n are not necessarily defined at the points in X^* . The set I_j consists of integers i with the property that the range of g_i is either equal to or properly contains the j th Y subinterval. The PDF for each Y subinterval depends on the PDF of X and the associated transformations g_i .

Theorem 4.1. Let X be a random variable of the continuous type with PDF $f_X(x)$ and with support \mathcal{X} , where \mathcal{X} consists of a finite union of open intervals. Let $g(x)$ be a real-valued function whose domain includes \mathcal{X} . Let $-\infty \leq x_1 < x_2 < \dots < x_n < x_{n+1} \leq +\infty$ be a sequence of extended real numbers which satisfy the following conditions:

1. The sequence includes the endpoints of the intervals whose union is \mathcal{X} .
2. The PDF $f_X(x)$ is continuous on each open interval $A_i = (x_i, x_{i+1})$ for $i = 1, 2, \dots, n$.
3. If $f_X(x)$ is not identically zero on A_i , then the function $g_i(x)$, which is the restriction of $g(x)$ to A_i , is monotone on A_i and has a nonzero derivative at each point in A_i , for $i = 1, 2, \dots, n$.

Let $X^* = \{x_1, x_2, \dots, x_{n+1}\}$.

Let $\alpha = \{i \mid f_X(x) \text{ is not identically zero on } A_i\}$.

Let $m_i = \min \left\{ \lim_{x \downarrow x_i} g(x), \lim_{x \uparrow x_{i+1}} g(x) \right\}$ for $i = 1, 2, \dots, n$.

Let $M_i = \max \left\{ \lim_{x \downarrow x_i} g(x), \lim_{x \uparrow x_{i+1}} g(x) \right\}$ for $i = 1, 2, \dots, n$.

Let $Y^* = \cup_{i \in \alpha} \{m_i, M_i\}$.

Let $m = \|Y^*\| - 1$, where $\|\cdot\|$ denotes cardinality.

Order the elements y_j of Y^* so that $y_1 < y_2 < \dots < y_{m+1}$.

Let $I_j = \{i \mid m_i \leq y_j \text{ and } y_{j+1} \leq M_i\}$, for $j = 1, 2, \dots, m$.

Then for $y \in (y_j, y_{j+1})$,

$$f_Y(y) = \sum_{i \in I_j} f_X \left(g_i^{-1}(y) \right) \left| \frac{dg_i^{-1}(y)}{dy} \right|$$

for $j = 1, 2, \dots, m$.

Proof. Without loss of generality, consider the j th Y subinterval (y_j, y_{j+1}) . Also suppose that a and b are any points that lie inside (y_j, y_{j+1}) such that $a < b$. Furthermore, let $M_i = \max\{g_i^{-1}(a), g_i^{-1}(b)\}$ and $m_i = \min\{g_i^{-1}(a), g_i^{-1}(b)\}$ for $i \in I_j$. Then $Y \in (a, b)$ if and only if X lies in (m_i, M_i) for some i in I_j , which implies that

$$P(a < Y < b) = \sum_{i \in I_j} P(m_i < X < M_i).$$

Since Y is a continuous random variable,

$$\begin{aligned}
 P(a < Y < b) &= \sum_{i \in I_j} \int_{m_i}^{M_i} f_X(x) \, dx \\
 &= \sum_{i \in I_j} \int_a^b f_X\left(g_i^{-1}(y)\right) \left| \frac{dg_i^{-1}(y)}{dy} \right| dy \\
 &= \int_a^b \sum_{i \in I_j} f_X\left(g_i^{-1}(y)\right) \left| \frac{dg_i^{-1}(y)}{dy} \right| dy
 \end{aligned}$$

where we performed the change-of-variable $y = g_i(x)$, or equivalently $x = g_i^{-1}(y)$, and have used the absolute value to treat both increasing and decreasing transformations (see Miller and Miller's proof of their Theorem 7.1 [62, pages 240–242]). Referring to their theorem, we see that the integrand

$$\sum_{i \in I_j} f_X\left(g_i^{-1}(y)\right) \left| \frac{dg_i^{-1}(y)}{dy} \right|$$

is the PDF for Y on the subinterval (y_j, y_{j+1}) . □

The theorem could have been stated more compactly. It is given in the form presented here in order to facilitate data structures and algorithms that will be implemented in the `APPL Transform` procedure.

4.2 Implementation in APPL

It could be quite tedious to implement this theorem by hand for large m and n . Since transformations of random variables are commonplace in applied probability, we wanted to include an implementation of the theorem in APPL's suite of procedures. Two significant implementation issues emerged. First, Maple may produce several candidates for g_i^{-1} , e.g., when $g_i(x) = x^2$. As alluded to earlier, Maple returns $g_i^{-1}(y) = -\sqrt{y}$ and $g_i^{-1}(y) = \sqrt{y}$. The correct inverse is selected by requiring that $g_i^{-1}(g_i(c_i)) = c_i$, where c_i is any point in the i th X subinterval. When the i th X subinterval has finite endpoints, c_i can be chosen to be the midpoint of the subinterval. In the cases where $x_1 = -\infty$ or $x_{n+1} = \infty$ [e.g., $X \sim N(\mu, \sigma^2)$], c_i must be selected more carefully. The algorithm for determining c_i is:

1. If $x_i = -\infty$ and $x_{i+1} = +\infty$, then $c_i = 0$.
2. If $x_i = -\infty$ and $x_{i+1} \neq +\infty$, then $c_i = x_{i+1} - 1$.
3. If $x_i \neq -\infty$ and $x_{i+1} = +\infty$, then $c_i = x_i + 1$.
4. For all other cases (i.e., x_i and x_{i+1} both finite), $c_i = \frac{x_i + x_{i+1}}{2}$.

The second implementation issue involves redundancies in lists. Maple doesn't recognize that the integer 3 and the floating-point value 3.0, for example, are redundant in a list. The `ReduceList` procedure finds all adjacent points that differ by a small prescribed threshold δ and discards redundant floating-point type elements.

The `Transform(X, g)` procedure has two arguments: X and g . The first argument is a random variable in the list-of-sublists format described in Chapter 3. The data structure for g is a "list-of-two-sublists" that is similar in nature to the first two sublists of X . Here we characterize g by listing its monotone components and the endpoints of the corresponding domains.

The following are some additional minor implementation issues that arose during the coding of the algorithm:

- The user must supply x_1, x_2, \dots, x_{n+1} . This ordered list consists of the endpoints of the open intervals that constitute \mathcal{X} and all locations in \mathcal{X} where $g(x)$ is discontinuous or non-differentiable, or $g'(x)$ is zero. A preprocessor to determine elements of X^* corresponding to $g'(x) = 0$ could be added to the algorithm if desired. Thus if the support of X is $(-1, 2)$, and $Y = g(X) = X^2$, this preprocessor would include $x = 0$ in X^* yielding $X^* = \{-1, 0, 2\}$.
- The partition points x_1, x_2, \dots, x_{n+1} must be chosen so that $f_X(x)$ is not defined in a piecewise fashion on any X subinterval (e.g., if X has a triangular distribution, the mode must be a partition point).
- Adding extra x -values in \mathcal{X} to X^* that are not maxima, minima, or saddle points of $g(x)$ or discontinuities of $f_X(x)$ or $g(x)$ will not affect the correctness of the algorithm's implementation of the theorem, although the implementation will be slower.
- Many transformations such as $g(x) = x + e^x$ do not have an inverse that can be expressed in a closed-form. Numerical methods could be used in conjunction with the algorithm to find individual values of $f_Y(y)$. We have not implemented this in our code. If $g(x) = x + e^x$, for example, then

$$\left| \frac{dg^{-1}(y)}{dy} \right| = \frac{1}{1 + e^x} \Big|_{x=g^{-1}(y)}.$$

- Periodic transformations can pose a problem in the implementation of the theorem. Maple uses the usual principal inverses for $\sin x$, $\cos x$, and $\tan x$, but the restricted ranges of these principal inverses are often inappropriate. When the algorithm tries to identify which inverse is appropriate, it is only given one choice, the principal inverse, but the domain of X might not coincide with the range of the principal inverse. Example 4.4 illustrates this problem and a circumvention.
- No error-trapping has been done to insure that $f_X(x)$ is a legitimate PDF. Also, there is limited error-trapping on $g(x)$ in that the procedure gives an error message when the inverse cannot be found.
- Symbolic parameters are allowed in $f_X(x)$ but not in $g(x)$.

4.3 Examples

This section contains some illustrative examples of using the algorithm described in the previous section to determine the distribution of $Y = g(X)$. Before using the program, we tested its performance on several common transformations. Given $X \sim N(0, 1)$ and $Y = g(X) = X^2$, the program correctly returned the PDF for a chi square random variable with one degree of freedom. Given $X \sim N(\mu, \sigma^2)$ and $Y = g(X) = (X - \mu)/\sigma$, the program correctly returned the PDF for a standard normal random variable. Now we will consider more complex examples that illustrate the theorem's implementation.

Example 4.1. Consider the first example introduced earlier: given the random variable $X \sim U(-1, 2)$, find the PDF of $Y = g(X) = X^2$.

Given X^* , the algorithm determines the relevant partitions of the supports of X and Y . Then it determines which X subintervals map onto which Y subintervals. The subintervals are apparent in Figure 4.1. Note the set X^* is displayed on the horizontal axis, Y^* on the vertical axis, both marked with the \times symbol. The transformation is partitioned into monotone segments (with identical or disjoint ranges) delineated by \bullet . The assignment of data structures for f and g and the call to **Transform** are as follows:

```
> X := UniformRV(-1, 2);
> g := [[x -> x ^ 2, x -> x ^ 2], [-infinity, 0, infinity]];
> Y := Transform(X, g);
```

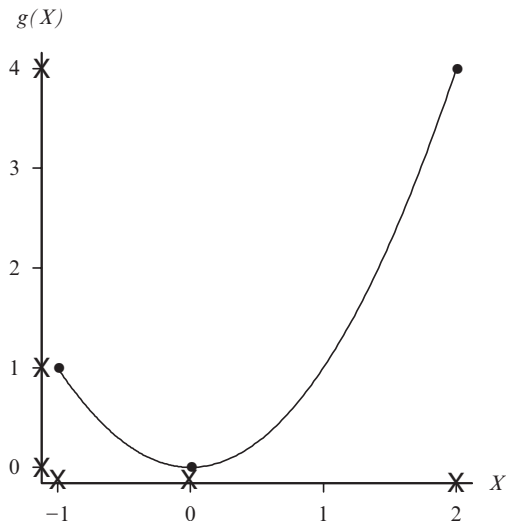


Fig. 4.1. The transformation $Y = g(X) = X^2$ for $-1 < X < 2$.

The **Transform** procedure determines that the transformation is 2-to-1 on $-1 < x < 1$ (excluding $x = 0$) and 1-to-1 on $1 \leq x < 2$. Since $Y = X^2$ has two inverses, the procedure determines which inverse to apply to which X subinterval. The resulting PDF for Y is

$$f_Y(y) = \begin{cases} 1/(3\sqrt{y}) & 0 < y < 1 \\ 1/(6\sqrt{y}) & 1 < y < 4. \end{cases}$$

The corresponding returned value of **Y** is a list-of-sublists that specifies the PDF of Y :

$$\left[\left[y \rightarrow \frac{1}{3\sqrt{y}}, y \rightarrow \frac{1}{6\sqrt{y}} \right], [0, 1, 4], [\text{"Continuous"}, \text{"PDF"}] \right]$$

The previous example could have been completely automated by letting Maple determine the extreme point, as opposed to specifying g separately on each of its monotone segments. The next example will use a nondifferentiable function to indicate why we allow the user to break a function into its monotone segments.

Example 4.2. Consider the random variable $X \sim U(0, 7)$. Find the PDF of $Y = g(X) = ||X - 3| - 1|$.

A graphical representation of this transformation is shown in Figure 4.2, with X^* , Y^* , and the monotone segments of g marked as before. This transformation is more complex than Example 4.1 in that it is 4-to-1 for $0 < y < 1$, 2-to-1 for $1 < y < 2$, and 1-to-1 for $2 < y < 3$. The APPL code

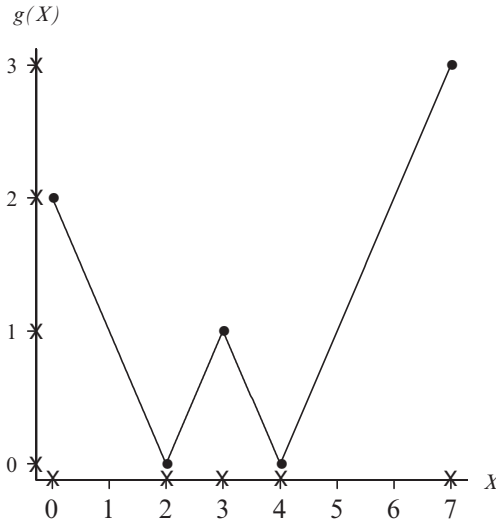


Fig. 4.2. The transformation $Y = g(X) = ||X - 3| - 1|$ for $0 < X < 7$.


```

> X := UniformRV(0, 7);
> g := [[x -> 2 - x, x -> x - 2, x -> 4 - x, x -> x - 4],
        [0, 2, 3, 4, 7]];
> Y := Transform(X, g);

```

yields the correct PDF for Y :

$$f_Y(y) = \begin{cases} 4/7 & 0 < y < 1 \\ 2/7 & 1 < y < 2 \\ 1/7 & 2 < y < 3. \end{cases}$$

Example 4.3. Consider the second example from beginning of the chapter: if the random variable X has PDF

$$f_X(x) = (x + 1)/18 \quad -1 < x < 5,$$

find the distribution of

$$Y = g(X) = \begin{cases} X^2 & -1 < X < \frac{3}{2} \\ X & \frac{3}{2} < X < 5. \end{cases}$$

The appropriate partition for the transformation with $n = 3$ and $m = 4$ is depicted in Figure 4.3. The APPL code for calculating the distribution of Y is

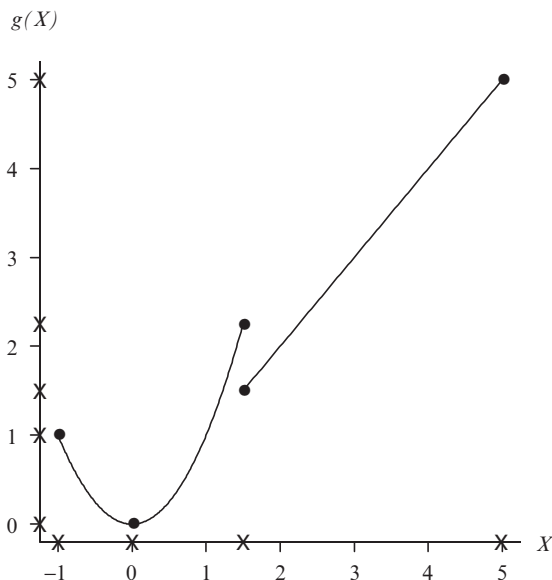


Fig. 4.3. The transformation $Y = g(X)$ has a discontinuity and is variously 2-to-1 and 1-to-1 on different subintervals.

```

> X := [[x -> (x + 1) / 18], [-1, 5],
        ["Continuous", "PDF"]];
> g := [[x -> x ^ 2, x -> x ^ 2, x -> x],
        [-1, 0, 3 / 2, 5]];
> Y := Transform(X, g);

```

The **Transform** procedure returns the following PDF for Y as a list-of-sublists:

$$f_Y(y) = \begin{cases} \frac{1}{18\sqrt{y}} & 0 < y < 1 \\ \frac{\sqrt{y}+1}{36\sqrt{y}} & 1 < y < 3/2 \\ \frac{(2y+3)\sqrt{y}+1}{36\sqrt{y}} & 3/2 < y < 9/4 \\ \frac{y+1}{18} & 9/4 < y < 5. \end{cases}$$

Example 4.4. As a final example, we will consider the problem that Casella and Berger [12] discussed, without providing $f_Y(y)$, as a prelude to their transformation theorem. Letting X be a uniform random variable on $(0, 2\pi)$, find the distribution of $Y = g(X) = \sin^2(X)$.

As Figure 4.4 shows, this transformation is 4-to-1 for the single subinterval for Y , which is $(0, 1)$. Furthermore, since $n = 4$, $X^* = \{0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi\}$. The PDF of Y is

$$f_Y(y) = \frac{1}{\pi\sqrt{y-y^2}} \quad 0 < y < 1,$$

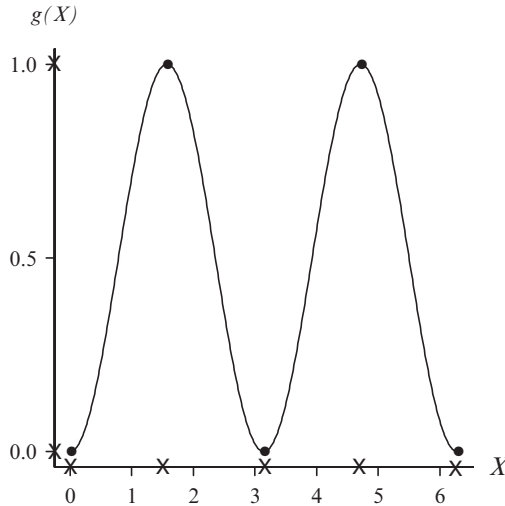


Fig. 4.4. The transformation $Y = g(X) = \sin^2(X)$ for $0 < X < 2\pi$.

which is commonly known as the arcsin distribution (Johnson, et al. [40, pages 253–255]). To overcome the principal inverse difficulty alluded to in the previous section, we devised the following equivalent problem: let the random variable $X \sim U(-\frac{\pi}{2}, \frac{\pi}{2})$ and let $Y = g(X) = \sin^2(X)$. In this case the domain of X will be the same as the range of g^{-1} which results from using the standard arcsin function. This solution yielded a random variable with a notable feature. The distribution's HF, $h_Y(y) = f_Y(y)/S_Y(y)$, might be of interest to a reliability engineer. For this distribution,

$$h_Y(y) = \frac{2}{\sqrt{y - y^2}[\pi - 2 \arcsin(2y - 1)]} \quad 0 < y < 1.$$

By plotting this HF, we see it has the rare property of a closed-form, “bathtub”-shaped HF. Furthermore, we can now apply the transformation $W = g(Y) = \lambda Y$ in order to derive a random variable W that is a one-parameter random variable with a closed-form, bathtub-shaped HF. To our knowledge, although the arcsin distribution has been discussed in the literature, we have yet to find mention of the fact that it has a bathtub-shaped HF. Most distributions with bathtub-shaped HFs must be analyzed numerically because their HFs are not closed form. One of the useful features of the **Transform** procedure is that it can help the practitioner gain insight into what transformations could result in a useful model. In this example, it becomes apparent that the transformation “crowds” the uniformly distributed X random variable into a distribution for Y that is “heavy” on either end of its support. The result is a random variable with a bathtub-shaped HF. The program enables the model designer to confirm, or gain insight into the univariate transformations that could result in a useful model.

The **Transform** procedure (Glen, et al. [28]) is one of the fundamental building blocks for APPL. Although the presentation in this chapter has been solely for continuous random variables, **Transform** is also able to find the distribution of a function of a discrete random variable. Applications of **Transform** will be postponed to the later chapters of this monograph.

Products of Random Variables

This chapter describes an algorithm for computing the PDF of the product of two independent continuous random variables. This algorithm has been implemented in the **Product** procedure in APPL. The algorithm behind the **Transform** procedure from the previous chapter differs fundamentally from the algorithm behind the **Product** procedure in that the former concerns the transformation of just *one* random variable and the latter concerns the product of *two* random variables. Some examples demonstrate the algorithm's application.

Rohatgi's well-known result [73, page 141] for determining the distribution of the product of two random variables is straightforward to derive, but difficult to implement: If X and Y are continuous random variables with joint PDF $f_{X,Y}(x, y)$, then the PDF of $V = XY$ is

$$f_V(v) = \int_{-\infty}^{\infty} f_{X,Y}\left(x, \frac{v}{x}\right) \frac{1}{|x|} dx.$$

Larsen and Marx [47, page 225] state the formula for the special case in which X and Y are independent. The implementation of this easily-stated result, however, is not straightforward for general X and Y . Springer [77] presents a chapter on finding distributions of products of random variables, relying mostly on Laplace and Mellin transformation techniques, as implementation of Rohatgi's result is often too cumbersome. Difficulties occur as a result of both the myriad variations to the limits of integration and the propensity of the PDF of V to be defined in a piecewise manner. In this chapter, we consider the cases when X and Y are independent and may have PDFs defined in a piecewise fashion. This simplification implies that the joint PDF of X and Y will be defined on a rectangular product space (Hogg, et al. [37, page 109]), which can be treated by considering special cases based on geometry.

5.1 Theorem

Considering a special case of Rohatgi's result will illustrate some of the issues associated with a general algorithm for determining the PDF of the product of two independent random variables. For completeness, the theorem below is proven from first principles (using the transformation technique) even though it could be stated as a special case of Rohatgi's result. Assume that the random variable X has support on the positive interval (a, b) and the random variable Y has support on the positive interval (c, d) . Also, the product space of the two random variables is assumed to fall entirely in the first quadrant. Theorems and proofs for other rectangular support regions are similar. The algorithm described in Section 5.3 includes all possible scenarios.

Theorem 5.1. Let X be a random variable of the continuous type with PDF $f_X(x)$ that is defined and positive on the interval (a, b) , where $0 < a < b < \infty$. Similarly, let Y be a random variable of the continuous type with PDF $f_Y(y)$ which is defined and positive on the interval (c, d) , where $0 < c < d < \infty$. The PDF of $V = XY$ is

$$f_V(v) = \begin{cases} \int_a^{v/c} f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & ac < v < ad \\ \int_{v/d}^{v/c} f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & ad < v < bc \\ \int_{v/d}^b f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & bc < v < bd \end{cases}$$

when $ad < bc$,

$$f_V(v) = \begin{cases} \int_a^{v/c} f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & ac < v < ad \\ \int_{v/d}^b f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & ad < v < bd \end{cases}$$

when $ad = bc$, and

$$f_V(v) = \begin{cases} \int_a^{v/c} f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & ac < v < bc \\ \int_a^b f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & bc < v < ad \\ \int_{v/d}^b f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & ad < v < bd \end{cases}$$

when $ad > bc$.

Proof. Only the case of $ad < bc$ is considered. The other cases are proven analogously. Using the transformation technique (Hogg et al. [37, page 87]), the transformation $Z = X$ and $V = XY$ constitutes a 1-1 mapping from $\mathcal{A} = \{(x, y) | a < x < b, c < y < d\}$ to $\mathcal{B} = \{(z, v) | a < z < b, cz < v < dz\}$. Let u denote the transformation and w the inverse transformation. The transformation, inverse, and Jacobian are

$$\begin{aligned} z &= u_1(x, y) = x & x &= w_1(z, v) = z \\ v &= u_2(x, y) = xy & y &= w_2(z, v) = v/z \end{aligned}$$

$$J = \begin{vmatrix} 1 & 0 \\ -v/z^2 & 1/z \end{vmatrix} = 1/z.$$

The joint PDF of Z and V is

$$f_{Z,V}(z, v) = f_X(w_1(z, v))f_Y(w_2(z, v)) |J|, \quad (z, v) \in \mathcal{B},$$

or, equivalently,

$$f_{Z,V}(z, v) = f_X(z)f_Y(v/z) \frac{1}{|z|}, \quad (z, v) \in \mathcal{B}.$$

Integrating with respect to z over the appropriate intervals and replacing z with x in the final result yields the marginal PDF of V ,

$$f_V(v) = \begin{cases} \int_a^{v/c} f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & ac < v < ad \\ \int_{v/d}^{v/c} f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & ad < v < bc \\ \int_{v/d}^b f_Y\left(\frac{v}{x}\right) f_X(x) \frac{1}{x} dx & bc < v < bd, \end{cases}$$

as desired. \square

The geometry associated with the transformation for the case $ad < bc$ considered in the proof is shown in Figures 5.1 and 5.2. The transformation maps A_i onto B_i for $i = 1, 2, 3$. Although the transformation technique has been used to prove this theorem, the CDF technique could also have been used. The theorem also follows directly from Rohatgi's result since

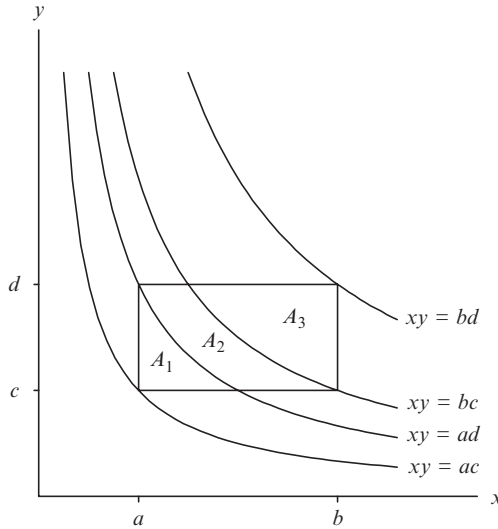


Fig. 5.1. The support of X and Y when $ad < bc$.

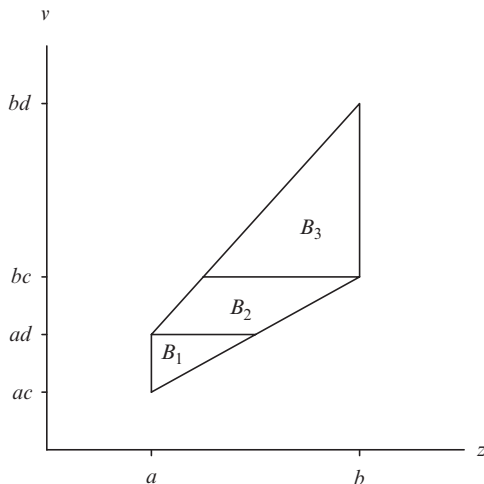


Fig. 5.2. The mapping of $Z = X$ and $V = XY$ when $ad < bc$.

$f_{X,Y}(x, v/x) = f_X(x)f_Y(v/x)$, which is nonzero only when $a < x < b$ and $c < v/x < d$. Thus $\max\{a, v/d\} < x < \min\{b, v/c\}$, which yields the limits of integration displayed in the theorem. The geometry of the case $ad < bc$ is shown in Figure 5.1 in which the regions A_1 , A_2 , and A_3 correspond to the three different integrals for that case. Each A_i corresponds to a different interval of v for the family of hyperbolas given by $xy = v$. Given v , the interval of integration consists of those values of x for which the curve $xy = v$ intersects the rectangle $\{(x, y) : a < x < b, c < y < d\}$.

5.2 Implementation in APPL

The theorem in the previous section illustrates the importance of considering the magnitudes of the product of the coordinates of the southeast and northwest corners of the product space [i.e., (b, c) and (a, d)] when it lies entirely in the first quadrant. In order to apply the theorem to any continuous random variables X and Y , three generalizations need to be addressed:

1. Analogous theorems must be written for the cases when the product space defined by (a, b) and (c, d) lies wholly in one of the other three quadrants. Rectangles that lie in more than one quadrant can be split into parts.
2. Instead of having PDFs that are specified by a single standard function over their entire support, the random variables X and Y may be defined in a piecewise manner over several intervals, forming many segments of the PDF (e.g., the triangular distribution).
3. The cases when 0 and/or $\pm\infty$ belong to the endpoints of the intervals which constitute the support of X and Y must be considered (e.g., the normal distribution, see Example 5.4).

These generalizations result in 24 (6 for each quadrant) primary cases that must be considered in order to correctly compute the limits of integration of the theorem. Further, some of these 24 cases must be broken into as many as three subcases, as illustrated in the theorem. In the first quadrant (i.e., $a \geq 0$ and $c \geq 0$), for example, we must consider the six primary cases:

1. $c = 0$ and $d = \infty$ (Scenario A, see the algorithm in Section 5.4),
2. $c = 0$ and $d < \infty$ (Scenario B),
3. $c > 0$ and $d = \infty$ (Scenario C),
4. $c > 0$ and $d < \infty$ (Scenario D, with Case I: $ad < bc$),
5. $c > 0$ and $d < \infty$ (Scenario D, with Case II: $ad = bc$), and
6. $c > 0$ and $d < \infty$ (Scenario D, with Case III: $ad > bc$).

The paragraphs below provide more details. For quadrants II, III and IV, the limits of integration must also be set based on the appropriate geometry, as in Figure 5.1.

For random variables that are defined in a piecewise fashion over various intervals, let n be the number of intervals for X and let m be the number of intervals for Y . There are mn rectangular product subspaces and the contribution of each to the value of the PDF of $V = XY$ must be computed. Furthermore, each product subspace can contribute differently to the PDF of V on up to three segments of the support of V (see Theorem 5.1). As a result, the PDF of V tends to become complicated very quickly, with an upper limit of $3mn$ segments to its PDF. Furthermore, each of these segments fits into one of the 24 cases of the integration mentioned above. For example, the product of two $U(0, 1)$ random variables (Scenario B from the list above) yields a random variable V with only one segment (see Example 5.1). But slightly changing X and Y to $X \sim U(1, 2)$ and $Y \sim U(3, 4)$ yields a $V = XY$ defined differently on three segments (see Example 5.2).

The case where the support of X or Y contains 0 [e.g., $U(-1, 2)$] poses special difficulty because some of the rectangular product spaces will not lie wholly in any one quadrant and cannot be handled by the previously developed techniques. Our solution to this difficulty is to artificially add 0 as one of the endpoints of the intervals for X and Y whenever this case occurs, producing redundant segments, i.e., a segment on each side of zero with the same formula for the PDF. This method allows for implementation when the support area intersects more than one quadrant. Example 5.4 gives an application of this special case, where the area of interest intersects all four quadrants. The algorithm subdivides the total area into four components, each entirely in one quadrant, and calculates the relevant portion of the new PDF for each quadrant.

The algorithm consists of a set-up portion, followed by nested loops that determine the contribution to the PDF of $V = XY$ separately for each of the four quadrants. Section 5.5 contains the set-up portion and the algorithm associated with the first quadrant. The algorithm for the other quadrants is similar.

The set-up phase begins by setting n and m to the number of intervals that form the support of X and Y . Next, 0 is added as an interval delimiter for X and/or Y if the random variable can assume both positive and negative values, and 0 is not already an interval delimiter. Finally, the endpoints of the intervals that form the support of V are determined by taking all products of the endpoints of the X intervals times the endpoints of the Y intervals.

A nested set of loops follows that treats all pairings of X and Y intervals. As shown in Figure 5.1, the coordinates (a, c) are assigned to the south-west corner of the current rectangle of interest, and the coordinates (b, d) are assigned to the northeast corner of the current rectangle of interest. A test to determine which quadrant contains the current rectangle is made at this point. Adding 0 as an interval delimiter in the set-up phase assures that the current rectangle will be completely contained in just one of the quadrants. Once the quadrant is determined, tests on c and d determine which integrals should be computed and the appropriate limits of integration. Finally, the insertion of 0 sometimes leads to a PDF for V with the same formula on both sides of 0. If this occurs, the program simplifies the PDF by removing 0 as an interval endpoint if the function is defined at 0.

When this algorithm is implemented in APPL, Maple occasionally returns a PDF in a less than optimal form. The expression $\ln(-v) - I\pi$, for example, may occur as part of a PDF. Since $\ln(-1) = I\pi$, the expression is equal to $\ln v$. Also the PDF may involve a complicated integral, which Maple fails to recognize is equal to a much simpler standard form (see Example 5.5). We anticipate that future generations of computer algebra systems will overcome these difficulties.

There are three classes of resultant distributions that come from products via this algorithm. Class A is the set of resultant distributions that are completely tractable, such as the product of two $U(0, 1)$ random variables (see Example 5.1). Here the PDF is in closed form, the CDF is calculable, quantiles can be calculated, and plots of distributions are easily produced. The second set, Class B, is the set of resultant distributions that are not in closed form, but are in forms that Maple can still evaluate, such as PDFs that rely on the `erf` function or the `BesselK` functions of Maple. Example 5.4 illustrates one such case, where the non-closed form PDF is expressed in terms of the `BesselK` function. In Class B, it is often possible to plot the PDF and calculate quantiles. Finding a usable form for the CDF, however, might not be possible. The final class of problems, Class C, is the set that only appears as un-evaluated integrals. The convolution of three Weibull random variables is one such example. Here the resulting PDF has little use, as Maple cannot plot the function, and cannot evaluate critical points very effectively.

5.3 Examples

This section contains examples of applying the APPL procedure `Product`, which is our implementation of the algorithm described in the previous section.

Example 5.1. Consider the independent random variables $X \sim U(0, 1)$ and $Y \sim U(0, 1)$. Find the distribution of $V = XY$.

This is a simple application of the algorithm. The following APPL code defines the distribution of the random variables X and Y and returns the PDF of their product:

```
> X := UniformRV(0, 1);
> Y := UniformRV(0, 1);
> V := Product(X, Y);
```

The resulting PDF for $V = XY$ is

$$f_V(v) = -\ln v \quad 0 < v < 1,$$

which can be readily verified by hand as follows: The CDF of V is

$$\begin{aligned} F_V(v) &= \Pr(V \leq v) \\ &= 1 - \Pr(V > v) \\ &= 1 - \Pr(XY > v) \\ &= 1 - \int_v^1 \int_{v/x}^1 1 \, dy \, dx \\ &= v - v \ln v \quad 0 < v < 1. \end{aligned}$$

Differentiating with respect to v yields the same PDF returned by APPL.

Example 5.1 illustrates the simplest case, where the PDF of V is defined on a single segment. The next example has a rectangular support for X and Y , which results in a piecewise PDF of V .

Example 5.2. Consider the independent random variables $X \sim U(1, 2)$ and $Y \sim U(3, 4)$. Find the distribution of $V = XY$.

The following APPL code defines the random variables X and Y and returns the PDF of their product.

```
> X := UniformRV(1, 2);
> Y := UniformRV(3, 4);
> V := Product(X, Y);
```

These statements yield the following PDF for $V = XY$:

$$f_V(v) = \begin{cases} \ln v - \ln 3 & 3 < v < 4 \\ \ln 4 - \ln 3 & 4 < v < 6 \\ 3 \ln 2 - \ln v & 6 < v < 8. \end{cases}$$

Note that while the PDF of both X and Y are defined on single segments that have positive interval limits, the PDF of V is defined

on three segments. All of the other APPL procedures described earlier can be applied to the random variable V . For instance, the first four moments of V can be calculated via the additional APPL statements

```
> Mean(V);
> Variance(V);
> Skewness(V);
> Kurtosis(V);
```

which return

$$\frac{21}{4} \qquad \frac{175}{144} \qquad \frac{54\sqrt{7}}{875} \qquad \frac{1647621}{765625}.$$

Examples 5.1 and 5.2 have defined X and Y using numeric parameters, with the joint support of X and Y being a bounded product space. The next example considers the distribution of the product of two random variables with symbolic parameters whose joint support is the entire first quadrant.

Example 5.3. Consider the independent random variables $X \sim \text{lognormal}(\mu_1, \sigma_1)$ and $Y \sim \text{lognormal}(\mu_2, \sigma_2)$. Show that the product of X and Y is also lognormal.

The APPL code to find the distribution of the product is

```
> X := LogNormalRV(mu1, sigma1);
> Y := LogNormalRV(mu2, sigma2);
> V := Product(X, Y);
```

Maple does not return the PDF of V in a form that can be directly identified as a lognormal PDF. But using the **Transform** procedure with transformation $g(V) = \ln(V)$ on the random variable V yields a random variable that is $N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. Because of the relationship between the lognormal and the normal distributions, we know that V is lognormal with PDF

$$f_V(v) = \frac{\exp(-(\ln v - \mu_1 - \mu_2)^2 / 2(\sigma_1^2 + \sigma_2^2))}{v\sqrt{2\pi(\sigma_1^2 + \sigma_2^2)}} \qquad 0 < v < \infty.$$

This can be identified as a $\text{lognormal}(\mu_1 + \mu_2, \sqrt{\sigma_1^2 + \sigma_2^2})$ random variable. Hence the product of two independent lognormal random variables is itself lognormal.

We now move outside the first quadrant to a distribution with a much less tractable distribution for $V = XY$.

Example 5.4. Consider the independent random variables $X \sim N(0, 1)$ and $Y \sim N(0, 1)$. Find the PDF of $V = XY$.

This example illustrates the case of 0 in the support of X and Y and also the case where the support of X and Y includes the endpoints $\pm\infty$. The APPL code to find the distribution of the product is

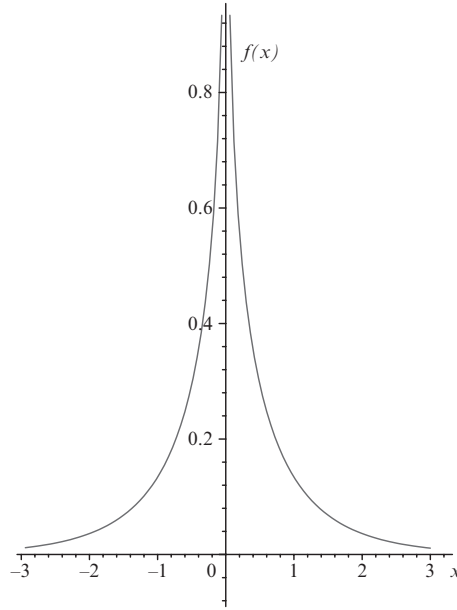


Fig. 5.3. The PDF of $V = XY$ for $X \sim N(0, 1)$ and $Y \sim N(0, 1)$.

```
> X := NormalRV(0, 1);
> Y := NormalRV(0, 1);
> V := Product(X, Y);
```

The `Product` procedure yields the following PDF for V :

$$f_V(v) = \begin{cases} \frac{\text{BesselK}(0, -v)}{\pi} & -\infty < v < 0 \\ \frac{\text{BesselK}(0, v)}{\pi} & 0 < v < \infty \end{cases}$$

which relies on Maple's `BesselK` function. An APPL plot of this function is given in Figure 5.3.

The following final, more complicated example involves checking the validity of a well-known random variate generation algorithm.

Example 5.5. Consider the independent random variables $U_1 \sim U(0, 1)$ and $U_2 \sim U(0, 1)$. The Box–Muller algorithm for generating a single standard normal deviate V can be coded in one line (see Devroye [16]) as

$$V \leftarrow \sqrt{-2 \ln U_1} \cos(2\pi U_2),$$

where U_1 and U_2 are independent random numbers. Use the **Transform** and **Product** procedures together to determine the PDF of V .

Due to the principle inverse difficulty with trigonometric functions (see Example 4.4 for an illustration), the transformation must be rewritten as

$$V \leftarrow \sqrt{-2 \ln U_1} \cos(\pi U_2)$$

before using **Transform**. The APPL code to compute the distribution of V is

```
> U1 := UniformRV(0, 1);
> U2 := UniformRV(0, Pi);
> T1 := Transform(U1, [[x -> sqrt(-2 * ln(x))], [0, 1]]);
> T2 := Transform(U2, [[x -> cos(x)], [0, Pi]]);
> V := Product(T1, T2);
```

This APPL code yields the following PDF for V :

$$f_V(v) = \begin{cases} \int_{-v}^{\infty} \frac{x e^{-x^2/2}}{\pi \sqrt{x^2 - v^2}} dx & -\infty < v < 0 \\ \int_v^{\infty} \frac{x e^{-x^2/2}}{\pi \sqrt{x^2 - v^2}} dx & 0 < v < \infty. \end{cases}$$

While this form is not easily recognizable as the PDF for the normal distribution, it is mathematically equivalent to the more standard

$$f_V(v) = \frac{1}{\sqrt{2\pi}} e^{-v^2/2} \quad -\infty < v < \infty.$$

5.4 Extensions

We consider two extensions to the **Product** procedure: (a) adapting the algorithm to automate finding the distribution of sums of random variables by using the **Product** and **Transform** procedures together, and (b) placing a loop around the **Product** code to compute the distribution of the product of several iid random variables.

First, by combining the algorithm from Chapter 4 for automating the transformations of random variables, it is possible to create an algorithm for finding the distribution of the sum of independent, univariate random variables. Note that the relationship $W = \ln(e^X e^Y)$ is equivalent to $W = X + Y$, when X and Y are independent random variables. This equivalence allows us to find the distributions of e^X and e^Y , then find the distribution of the product of the two, then find the distribution of the natural logarithm of this product to yield the distribution of the sum of X and Y . So APPL

includes the **Convolution** procedure, which is used to find the distribution of the sum of X and Y . Likewise, there is an ancillary APPL procedure **Difference** that uses **Transform** and **Product** to find the distribution of $X - Y$. Additionally, **Transform** and **Product** can be used together to find the distribution of X/Y . For example, the single line of APPL code

```
> Convolution(UniformRV(0, 1), UniformRV(0, 1));
```

calculates the distribution of the sum of two independent $U(0, 1)$ random variables in the fashion described above.

Second, applications might arise when it is of interest to find the distribution of the product of several iid random variables. For instance, if W is the product of eight $U(0, 1)$ random variables, the distribution of W can be found using the following APPL code:

```
> X := UniformRV(0, 1);
> Y := Product(X, X);
> Z := Product(Y, Y);
> W := Product(Z, Z);
```

The program is clearer, and more easily generalized if a Maple **for** loop is used:

```
> X := UniformRV(0, 1);
> W := Product(X, X);
> for i from 3 to 8 do
>   W := Product(W, X);
> od;
```

APPL contains a procedure named **ProductIID** which automates this process, giving a more readable code segment than the two above:

```
> X := UniformRV(0, 1);
> W := ProductIID(X, 8);
```

or, equivalently, but more succinctly,

```
> W := ProductIID(UniformRV(0, 1), 8);
```

Since the **Product** procedure is fairly short, we end this chapter by presenting the **Product** algorithm in pseudocode.

5.5 Algorithm

This section gives the details associated with the **Product** algorithm, which computes the PDF of the product of two continuous, independent random variables. Cardinality is denoted by $\|\cdot\|$ and the empty list is denoted by $[]$. Indentation is used to indicate loops and conditions. Comments are set flushright. To allow subscripting, the PDFs $f_X(x)$, $f_Y(y)$, and $f_V(v)$ are replaced by $f(x)$, $g(y)$, and $h(v)$. The conditions prior to the calculation of the integrals are imposed to prevent calculating integrals unnecessarily.

Procedure Product: Find the PDF of the product of X and Y .

Input: The distribution of the random variable X and the distribution of the random variable Y , where X and Y are independent, univariate, continuous random variables. The PDFs are in the list-of-sublists format, i.e., the random variable X is represented by the data structure $[\tilde{f}(x), x^*, ["Continuous", "PDF"]]$, and the random variable Y is represented by the data structure $[\tilde{g}(y), y^*, ["Continuous", "PDF"]]$. The first sublist for X is a list of the piecewise $f(x)$ functions, that is, $\tilde{f}(x) = [f_1(x), f_2(x), \dots, f_n(x)]$. The second sublist for X is a list of the endpoints of the support of X , that is, $x^* = [x_1^*, x_2^*, \dots, x_{n+1}^*]$, where $f(x) = f_i(x)$ for $x_i^* < x < x_{i+1}^*$, $i = 1, 2, \dots, n$.

Output: The PDF of $V = XY$, $h(v)$, in the same list-of-sublists format as the input. The list $\tilde{h}(v) = [h_1(v), h_2(v), \dots, h_l(v)]$ contains the piecewise elements of the PDF of V and $v^* = [v_1^*, v_2^*, \dots, v_{l+1}^*]$ contains the corresponding breakpoints of the support of V . The PDF is determined by a special case of Rohatgi's result for two independent random variables:

$$h(v) = \int_{-\infty}^{\infty} f(x) \cdot g\left(\frac{v}{x}\right) \frac{1}{|x|} dx.$$

Algorithm:

```

 $n \leftarrow \|x^*\| - 1$                                  $n$  is the number of segments of the PDF of  $X$ 
 $m \leftarrow \|y^*\| - 1$                                  $m$  is the number of segments of the PDF of  $Y$ 
if ( $x_1^* < 0$  and  $x_{n+1}^* > 0$  and  $0 \notin x^*$ ) then      insert 0 into  $x^*$  if necessary
  for  $i \leftarrow 1$  to  $n$ 
    if ( $x_i^* < 0$  and  $x_{i+1}^* > 0$ ) then
      insert 0 between positions  $x_i^*$  and  $x_{i+1}^*$ 
      insert  $f_i(x)$  after element  $i$  in the list  $\tilde{f}(x)$ 
       $n \leftarrow n + 1$ 
    break
if ( $y_1^* < 0$  and  $y_{m+1}^* > 0$  and  $0 \notin y^*$ ) then    insert 0 into  $y^*$  if necessary
  for  $i \leftarrow 1$  to  $m$ 
    if ( $y_i^* < 0$  and  $y_{i+1}^* > 0$ ) then
      insert 0 between positions  $y_i^*$  and  $y_{i+1}^*$ 
      insert  $g_i(y)$  after element  $i$  in the list  $\tilde{g}(y)$ 
       $m \leftarrow m + 1$ 
    break
 $v^* \leftarrow []$                                        initialize the support list for  $V$ 
for  $i \leftarrow 1$  to  $n + 1$                                build the support list  $v^*$ 
  for  $j \leftarrow 1$  to  $m + 1$ 
     $v^* \leftarrow v^* \cup (x_i^* \cdot y_j^*)$ 
 $v^* \leftarrow \text{sort}(v^*)$                                order the endpoints of the support of  $V$ 
 $l \leftarrow \|v^*\| - 1$                                 $l$  is the number of segments of the PDF of  $V$ 
for  $i \leftarrow 1$  to  $l$                                initialize PDF segments of  $V$ 

```

```

 $h_i(v) \leftarrow 0$ 
for  $i \leftarrow 1$  to  $n$                                 loop through each piecewise segment of  $X$ 
  for  $j \leftarrow 1$  to  $m$                                 loop through each piecewise segment of  $Y$ 
     $a \leftarrow x_i^*$                                 define corners of the rectangular region
     $b \leftarrow x_{i+1}^*$ 
     $c \leftarrow y_j^*$ 
     $d \leftarrow y_{j+1}^*$ 
    if  $(a \geq 0$  and  $c \geq 0)$  then                    the region is in the first quadrant
       $p(v) \leftarrow \int_a^b f_i(x) \cdot g_j(\frac{v}{x}) \cdot \frac{1}{x} dx$     compute required integrals
      if  $(d < \infty)$  then  $q(v) \leftarrow \int_{v/d}^b f_i(x) \cdot g_j(\frac{v}{x}) \cdot \frac{1}{x} dx$ 
      if  $(c > 0)$  then  $r(v) \leftarrow \int_a^{v/c} f_i(x) \cdot g_j(\frac{v}{x}) \cdot \frac{1}{x} dx$ 
      if  $(c > 0$  and  $d < \infty$  and  $ad < bc)$  then
         $s(v) \leftarrow \int_{v/d}^{v/c} f_i(x) \cdot g_j(\frac{v}{x}) \cdot \frac{1}{x} dx$ 
      if  $(c = 0$  and  $d = \infty)$  then                Scenario A from Section 5.2
        for  $k \leftarrow 1$  to  $l$ 
          if  $(v_k^* \geq 0)$  then  $h_k(v) \leftarrow h_k(v) + p(v)$ 
      if  $(c = 0$  and  $d < \infty)$  then                Scenario B from Section 5.2
        for  $k \leftarrow 1$  to  $l$ 
          if  $(v_k^* \geq 0$  and  $v_{k+1}^* \leq ad)$  then  $h_k(v) \leftarrow h_k(v) + p(v)$ 
          if  $(v_k^* \geq ad$  and  $v_{k+1}^* \leq bd)$  then  $h_k(v) \leftarrow h_k(v) + q(v)$ 
      if  $(c > 0$  and  $d = \infty)$  then                Scenario C from Section 5.2
        for  $k \leftarrow 1$  to  $l$ 
          if  $(v_k^* \geq bc)$  then  $h_k(v) \leftarrow h_k(v) + p(v)$ 
          if  $(v_k^* \geq ac$  and  $v_{k+1}^* \leq bc)$  then  $h_k(v) \leftarrow h_k(v) + r(v)$ 
      if  $(c > 0$  and  $d < \infty)$  then                Scenario D from Section 5.2
        if  $(ad < bc)$  then                            Case I
          for  $k \leftarrow 1$  to  $l$ 
            if  $(v_k^* \geq ac$  and  $v_{k+1}^* \leq ad)$  then  $h_k(v) \leftarrow h_k(v) + r(v)$ 
            if  $(v_k^* \geq ad$  and  $v_{k+1}^* \leq bc)$  then  $h_k(v) \leftarrow h_k(v) + s(v)$ 
            if  $(v_k^* \geq bc$  and  $v_{k+1}^* \leq bd)$  then  $h_k(v) \leftarrow h_k(v) + q(v)$ 
          if  $(ad = bc)$  then                            Case II
            for  $k \leftarrow 1$  to  $l$ 
              if  $(v_k^* \geq ac$  and  $v_{k+1}^* \leq ad)$  then  $h_k(v) \leftarrow h_k(v) + r(v)$ 
              if  $(v_k^* \geq bc$  and  $v_{k+1}^* \leq bd)$  then  $h_k(v) \leftarrow h_k(v) + q(v)$ 
          if  $(ad > bc)$  then                            Case III
            for  $k \leftarrow 1$  to  $l$ 
              if  $(v_k^* \geq ac$  and  $v_{k+1}^* \leq bc)$  then  $h_k(v) \leftarrow h_k(v) + r(v)$ 
              if  $(v_k^* \geq bc$  and  $v_{k+1}^* \leq ad)$  then  $h_k(v) \leftarrow h_k(v) + p(v)$ 
              if  $(v_k^* \geq ad$  and  $v_{k+1}^* \leq bd)$  then  $h_k(v) \leftarrow h_k(v) + q(v)$ 
        analogous steps for rectangles in quadrants II, III, and IV
    remove any redundancies in  $f_V(v)$  due to 0 in the support list
  return( $[\tilde{h}(v), v^*, ["Continuous", "PDF"]]$ )

```


In conclusion, an algorithm for calculating the PDF of the product of two independent random variables X and Y , as detailed in Glen et al. [30], has been developed and implemented. Special care has been taken to allow for random variables that are defined in a piecewise manner or have 0 as a part of their support. The algorithm can be extended to calculate the PDF of sums of independent random variables and to products of iid random variables. Although its presentation in this chapter has been solely for continuous random variables, the algorithm is also able to find the distribution of the product of two discrete random variables. This concludes our introduction to continuous random variables. The next three chapters are devoted to discrete random variables.

Data Structures and Simple Algorithms

This chapter introduces the data structures necessary to define a discrete random variable in APPL and surveys some simple algorithms associated with discrete random variables. The first section will show that the nature of the support of discrete random variables makes the data structures required much more complicated than for continuous random variables.

6.1 Data Structures

Recall from Chapter 3 that the format for describing a continuous random variable is a Maple list consisting of three sublists:

$$[[f(x)], [support], ["Continuous", "XXX"]]$$

where XXX is PDF, CDF, SF, HF, CHF, or IDF. It might seem like a trivial matter to simply modify the list-of-sublists to

$$[[f(x)], [support], ["Discrete", "XXX"]]$$

but there is a complication. The problem lies in the second sublist—the support. For example, let X denote the number of heads in two tosses of a fair coin. The binomial distribution with $n = 2$ and $p = 1/2$ is an appropriate probability model for X . The APPL statement

```
> X := [[1 / 4, 1 / 2, 1 / 4], [0, 1, 2], ["Discrete", "PDF"]];
```

defines X using the above format for a discrete random variable. But what if $n = 1000$? Keying 1001 support values into the second sublist is impractical. The data structure must provide a way to automate the input when the support contains many values. Furthermore, what if we are interested in finding the distribution of $Y = X^2$ by using the **Transform** procedure? There also needs to be a mechanism available for easily encoding the support $1, 4, 9, \dots, n^2$.

The acronyms that identify the functional form that defines the distribution of a discrete random variable X in the second element of the third sublist are the same as in the continuous case:

- probability density function (PDF) $f_X(x) = \Pr(X = x)$,
- cumulative distribution function (CDF) $F_X(x) = \sum_{s \leq x} f_X(s)$,
- survivor function (SF) $S_X(x) = \sum_{s \geq x} f_X(s)$,
- hazard function (HF) $h_X(x) = f_X(x)/S_X(x)$,
- cumulative hazard function (CHF) $H_X(x) = -\log(S_X(x))$, and
- inverse distribution function (IDF) $F_X^{-1}(x)$.

The probability density function is also commonly referred to as the *probability mass function*.

The common data structure used in APPL is again referred to as a “list-of-sublists.” Each APPL random variable is input as a list that contains three sublists, each with a specific purpose. The first sublist contains either a function or a numeric list that defines one of the six functional forms that describe the distribution. For example, the PDF form of the Poisson distribution with mean λ and support $x = 0, 1, 2, \dots$ has as its first sublist

$$\left[x \rightarrow \frac{\lambda^x e^{-\lambda}}{x!} \right]$$

The CDF form of the geometric($1/4$) distribution with support $x = 1, 2, \dots$, has as its first sublist

$$\left[x \rightarrow 1 - \left(\frac{3}{4} \right)^x \right]$$

The SF form (in a numeric Maple list) of the probability of obtaining an x or higher, where $x = 1, 2, \dots, 6$, on a single roll of a fair 6-sided die is

$$\left[1, \frac{5}{6}, \frac{2}{3}, \frac{1}{2}, \frac{1}{3}, \frac{1}{6} \right]$$

The second sublist contains the random variable’s support. Recall from Chapter 3 that for a continuous random variable, this second sublist contains an ordered list of real numbers that delineate the end points of the intervals for the functions in the first sublist. The end point of each interval is automatically the starting point of the subsequent interval. A discrete random variable’s support, on the other hand, can be input in one of several different formats. This variation in formats presents greater difficulty than in the continuous case for determining a structure for the second sublist. For example, the Poisson distribution with PDF

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad x = 0, 1, 2, \dots$$

has as its support the nonnegative integers. This support has a pattern to it; the first value of x at which the PDF is defined is zero and the rest of the

support consists of subsequent integers. Many discrete random variables do not have a patterned support. When designing the second sublist structure for discrete random variables, we first had to distinguish between random variables that had some type of pattern to their support versus those that did not. For example, let X be a binomial random variable with parameters $n = 5$ and $p = 0.2$, and associated PDF

$$f(x) = \binom{5}{x} (0.2)^x (0.8)^{5-x} \quad x = 0, 1, \dots, 5.$$

The support of the random variable X consists of consecutive integers. For such random variables, only the first and last values of the support are needed to generate the entire support list. We call this support case the *Dot* case, since we can write the support in Maple's range (also called type '..') format: $\min\{\Omega\} \dots \max\{\Omega\}$, where Ω is the support. Thus, the second sublist for the random variable X is input as $[0 \dots 5]$.

Switching to the *NoDot* case, consider the random variable Y with PDF

$$f(y) = \begin{cases} 0.2 & y = 1 \\ 0.5 & y = 7/2 \\ 0.3 & y = 11. \end{cases}$$

Random variables with supports that display no pattern, such as the support of Y , must be entered as a Maple list, where the list elements are separated by commas. We call this support case the *NoDot* case. The support values listed in sublist two correspond to the distribution's function values in the first sublist. We input the first and second sublists of Y as $[0.2, 0.5, 0.3]$ and $[1, 7 / 2, 11]$, respectively.

After distinguishing between the *Dot* and *NoDot* support cases, there are subcases of these two general cases to consider. First, in the *NoDot* case, the function in the first sublist can be written as a formula or a list of numeric elements. The random variable Y with PDF values $[0.2, 0.5, 0.3]$ is a Maple list of numeric elements (separated by commas). On the other hand, the random variable X with PDF

$$f(x) = \frac{x}{27} \quad x = 1, 3, 7, 16,$$

is a valid discrete probability mass function whose PDF can be written as a formula in Maple's function notation as $x \rightarrow x/27$. Its first two sublists are input in Maple as $[x \rightarrow x / 27]$ and $[1, 3, 7, 16]$. APPL allows the user to enter a discrete random variable represented in the *NoDot* case in either format, numeric or formulaic. Converting the first sublist from the formulaic *NoDot* format to a "standard" *NoDot* format (where the first sublist is *not* written as a formula) is handled in a procedure called **Convert**. If we enter X as displayed above and apply the **Convert** procedure to X with the APPL statements

```
> X := [[x -> x / 27], [1, 3, 7, 16], ["Discrete", "PDF"]];
> X := Convert(X);
```

the random variable X is returned as the list

$$\left[\left[\frac{1}{27}, \frac{1}{9}, \frac{7}{27}, \frac{16}{27} \right], [1, 3, 7, 16], ["Discrete", "PDF"] \right]$$

More examples of the **Convert** procedure are given subsequently.

The *Dot* case also has two subcases: either the random variable has finite or infinite support. Let Ω represent the support of a random variable X . The “standard” *Dot* format for defining the support has the general form

$$[\min\{\Omega\} \dots \max\{\Omega\}, k, x \rightarrow g(x)]$$

where the default value of the support increment k (if not entered) is 1 and the default value of support transformation function $g(x)$ (if not entered) is the identity function.

To introduce the different variations in format, let $X_1 \sim \text{geometric}(1/4)$ with PDF

$$f_{X_1}(x) = \frac{1}{4} \left(\frac{3}{4} \right)^{x-1} \quad x = 1, 2, \dots$$

Let $X_2 = 2Y$, where $Y \sim \text{binomial}(5, 0.2)$. The PDF of X_2 is

$$f_{X_2}(x) = \binom{5}{x/2} (0.2)^{x/2} (0.8)^{5-x/2} \quad x = 0, 2, 4, \dots, 10.$$

Let X_3 be a discrete random variable with PDF

$$f_{X_3}(x) = \frac{x+6}{54} \quad x = 1, 4, 9, 16.$$

Let X_4 be a discrete random variable with PDF

$$f_{X_4}(x) = \frac{\sqrt{x}}{15} \quad x = 9, 25, 49.$$

The support of each of these random variables has a pattern that can be accounted for in the *Dot* data structure. X_1 ’s support is incremented by ones, while X_2 ’s support is incremented by twos. The support of X_3 is not only incremented by ones, but is also transformed by the function $g(x) = x^2$. Going a step further, X_4 ’s support is incremented by twos starting with $x = 3$ and then transformed by $g(x) = x^2$. In each of these cases, the minimum and maximum values of the support, the increment of the support, and the type of transformation applied to the support is pertinent information. Thus, in the *Dot* case, the second sublist will consist of either 1, 2, or 3 elements separated by commas containing this information. This general capability has been included in the data structure to accommodate discrete algorithmic procedures, such as **Transform**.

The supports of the random variables X_1 , X_2 , X_3 , and X_4 are input in the second sublist in the APPL list-of-sublists as

$$\begin{aligned} X_1 &: [1 \dots \infty] \\ X_2 &: [0 \dots 10, 2] \\ X_3 &: [1 \dots 4, x \rightarrow x^2] \\ X_4 &: [3 \dots 7, 2, x \rightarrow x^2] \end{aligned}$$

In summary, if the random variable X is discrete, its support will match one of the cases displayed in Table 6.1. An example of the APPL list-of-sublists format of a random variable from each of these cases is given in the outline that immediately follows Table 6.1.

1. *NoDot* SUPPORT FORMAT: The random variable's support Ω is a Maple numeric list of elements.
 - a) NUMERIC PDF: The random variable's PDF is a Maple numeric list of elements.

Example 6.1. Let X be the random variable with PDF

$$f(x) = \begin{cases} 0.6 & x = 2.5 \\ 0.3 & x = 3 \\ 0.1 & x = 15. \end{cases}$$

Table 6.1. Discrete random variable support categories for various types of support, support increments k , and support transformation functions $g(x)$.

Support Ω	Cases	Subcases		Examples
<i>NoDot</i> $\Omega = [x_1, x_2, \dots, x_n]$	Numeric PDF	—		6.1
	Formulaic PDF			6.2
<i>Dot</i> $\Omega = [\min\{\Omega\} \dots \max\{\Omega\},$ $k, x \rightarrow g(x)]$	Finite Support	$k = 1$	$g(x) = x$	6.3
			$g(x) \neq x$	6.4
		$k \neq 1$	$g(x) = x$	6.5
			$g(x) \neq x$	6.6
	Infinite Support	$k = 1$	$g(x) = x$	6.7
			$g(x) \neq x$	6.8
		$k \neq 1$	$g(x) = x$	6.9
			$g(x) \neq x$	6.10

This random variable X is input in APPL as

```
> X := [[0.6, 0.3, 0.1], [2.5, 3, 15],
        ["Discrete", "PDF"]];
```

- b) FORMULAIC PDF: The random variable's PDF is formulaic.

Example 6.2. Let X be the random variable with PDF

$$f(x) = x/8 \quad x = 1, 3, 4.$$

This random variable X is input in APPL as

```
> X := [[x -> x / 8], [1, 3, 4], ["Discrete", "PDF"]];
```

2. *Dot* SUPPORT FORMAT: The random variable's support Ω is of Maple type “..”, i.e., $\min\{\Omega\} \dots \max\{\Omega\}$.

- a) FINITE SUPPORT: The random variable's support is finite.

- i. The random variable's support is incremented by $k = 1$ and transformed by the identity function, i.e., $g(x) = x$ for all x .

Example 6.3. Let X be a Benford random variable (Benford [8]) with PDF

$$f(x) = \log_{10}(1 + 1/x) \quad x = 1, 2, \dots, 9.$$

The Benford random variable is input in APPL as

```
> X := [[x -> log[10](1 + 1 / x)], [1 .. 9],
        ["Discrete", "PDF"]];
```

- ii. The random variable's support is incremented by $k = 1$ and transformed by a function other than the identity function, i.e., $g(x) \neq x$ for some or all x .

Example 6.4. Let X be a random variable with PDF

$$f(x) = x/224 \quad x = 8, 27, 64, 125.$$

This random variable is input in APPL as

```
> X := [[x -> x / 224], [2 .. 5, x -> x ^ 3],
        ["Discrete", "PDF"]];
```

- iii. The random variable's support is incremented by $k \neq 1$ and transformed by $g(x) = x$.

Example 6.5. Let $Y \sim \text{binomial}(5, p)$, and let $X = 2Y$. This random variable X is input in APPL as

```
> X := [[x -> 120 * p ^ (x / 2) *
          (1 - p) ^ (5 - x / 2) /
          ((5 - x / 2)! * (x / 2)!)],
         [0 .. 10, 2], ["Discrete", "PDF"]];
```

- iv. The random variable's support is incremented by $k \neq 1$ and transformed by $g(x) \neq x$.

Example 6.6. Let X be a random variable with PDF

$$f(x) = 1600/489x \quad x = 4, 25, 64.$$

This random variable is input in APPL as

```
> X := [[x -> 1600 / (489 * x)],
         [2 .. 8, 3, x -> x ^ 2],
         ["Discrete", "PDF"]];
```

- b) INFINITE SUPPORT: The random variable's support is infinite.
- i. The random variable's support is incremented by $k = 1$ and transformed by $g(x) = x$.

Example 6.7. Let X be a negative binomial random variable with parameters $r = 3$ and $p = 1/3$. The simplified PDF of X is

$$f(x) = \frac{(x-2)(x-1)}{16} \left(\frac{2}{3}\right)^x \quad x = 3, 4, \dots$$

This random variable is input in APPL as

```
> X := [[x -> ((x - 2)*(x - 1)) / 16)*(2 / 3) ^ x],
         [3 .. infinity], ["Discrete", "PDF"]];
```

- ii. The random variable's support is incremented by $k = 1$ and transformed by $g(x) \neq x$.

Example 6.8. Let X be the square of a geometric($\frac{1}{2}$) random variable; i.e.,

$$f(x) = 1/2\sqrt{x} \quad x = 1, 4, 9, \dots$$

This random variable is input in APPL as

```
> X := [[x -> 2 ^ (-sqrt(x))],
         [1 .. infinity, x -> x ^ 2],
         ["Discrete", "PDF"]];
```

- iii. The random variable's support is incremented by $k \neq 1$ and transformed by $g(x) = x$.

Example 6.9. Let $Y \sim \text{geometric}(\frac{1}{2})$. Let $X = 2Y$, which has PDF

$$f(x) = \left(\frac{1}{2}\right)^{(x/2)} \quad x = 2, 4, 6, \dots$$

This random variable is input in APPL as

```
> X := [[x -> (1 / 2) ^ (x / 2)], [2 .. infinity, 2],
        ["Discrete", "PDF"]];
```

- iv. The random variable's support is incremented by $k \neq 1$ and transformed by $g(x) \neq x$.

Example 6.10. Let $Y \sim \text{geometric}(\frac{1}{2})$ and $X = 4Y^2$. The PDF of X is

$$f(x) = \left(\frac{1}{2}\right)^{\sqrt{x}/2} \quad x = 4, 16, 36, \dots$$

This random variable is input in APPL as

```
> X := [[x -> (1 / 2) ^ (sqrt(x) / 2)],
        [2 .. infinity, 2, x -> x ^ 2],
        ["Discrete", "PDF"]];
```

Whenever an APPL procedure is invoked to perform an operation on a discrete random variable, the random variable is first converted to its standard format within the procedure. As discussed in the preceding section, there is a standard discrete data structure format for both the *NoDot* and *Dot* cases. The conversion is necessary so that the main body of each procedure can avoid the complexity resulting from multiple formats.

Let X , for example, be the random variable discussed earlier in this chapter with PDF

$$f(x) = \frac{x}{27} \quad x = 1, 3, 7, 16.$$

Suppose the random variable X is input in APPL as

```
> X := [[x -> x / 27], [1, 3, 7, 16], ["Discrete", "PDF"]];
```

When an APPL procedure, such as **Mean**, receives the random variable **X** as an argument, it first determines if **X** has a *Dot* or *NoDot* support format. Since the support $\Omega = [1, 3, 7, 16]$ has a *NoDot* support format, the procedure's formula for computing the mean expects that it is receiving the random variable **X** in its converted format, which is

$$\left[\left[\frac{1}{27}, \frac{1}{9}, \frac{7}{27}, \frac{16}{27} \right], [1, 3, 7, 16], ["Discrete", "PDF"] \right]$$

In this format, the mean of a random variable with support values x_1, x_2, \dots with a *NoDot* support format is computed as

$$\begin{aligned}
E[X] &= \sum_{i=1}^{|\Omega|} x_i \cdot f(x_i) \\
&= \sum_{i=1}^4 \mathbf{x}[2][i] \cdot \mathbf{x}[1][i] \\
&= 1 \cdot \frac{1}{27} + 3 \cdot \frac{1}{9} + 7 \cdot \frac{7}{27} + 16 \cdot \frac{16}{27} \\
&= \frac{35}{3}.
\end{aligned}$$

where $\mathbf{x}[2][i]$ is the i th element in sublist two (X 's support) and $\mathbf{x}[1][i]$ is the i th element in sublist one (X 's probability values).

Again, each APPL procedure operates on random variables only in the standard *NoDot* and *Dot* formats. Each APPL procedure internally converts arguments to one of the two pre-defined formats, which allows algorithms to be developed that exploit these formats. Without these standard formats, an APPL procedure would be forced to discern the exact form of each random variable it encountered. The computations required in the procedure would then depend on the unique structure of each random variable, and as indicated in Table 6.1, there are ten acceptable formats for a discrete random variable in APPL. The **Convert** procedure provides each APPL procedure with a standard structural format for a discrete random variable.

Some random variables with finite support, such as the random variable Y with PDF

$$f(y) = \frac{y}{15} \quad y = 3, 5, 7,$$

can be input in APPL in several different formats, which include both a *NoDot* and *Dot* format. The APPL statements

```

> Y := [[1 / 5, 1 / 3, 7 / 15], [3, 5, 7], ["Discrete", "PDF"]];
> Y := [[y -> y / 15], [3, 5, 7], ["Discrete", "PDF"]];
> Y := [[y -> y / 15], [3 .. 7, 2], ["Discrete", "PDF"]];

```

define the same random variable Y in APPL. The first two lines define Y in its *NoDot* formats, while the third line displays Y in one possible *Dot* format. Although Y may be defined by using any of the several different formats, any algorithm or formula applied to Y will have the same outcome. Conversely, a discrete random variable with countably infinite support can only be input using a *Dot* support format. Since it is impossible to physically list every element of a countably infinite set, the *NoDot* support format for this type of random variable cannot be used.

The **Convert** procedure acts on discrete random variables in both the *Dot* and *NoDot* cases. It converts a discrete random variable X with a

- *Dot* support format to the standard APPL *Dot* support format. If Ω is the support of X , then the standard *Dot* second sublist support format is

$$[\min\{\Omega\} \dots \max\{\Omega\}, k, x \rightarrow g(x)]$$

where k is the support increment and $g(x)$ is the support transformation function.

- *NoDot* support format to the standard APPL *NoDot* support format with corresponding PDF. If $f(x)$ is the PDF and the support is $x = x_1, x_2, \dots, x_n$, then the PDF first sublist and the standard *NoDot* second sublist support format are

$$[f(x_1), f(x_2), \dots, f(x_n)], [x_1, x_2, \dots, x_n]$$

Two examples of the **Convert** procedure for the *NoDot* and *Dot* formats are included for further clarification.

Example 6.11. Use the APPL **Convert** procedure to convert a Zipf random variable (Ross [75, page 182]) with parameter $\alpha = 1$ to the standard *Dot* format.

The APPL statements below define X as the desired Zipf random variable and convert it to its standard *Dot* format.

```
> X := [[x -> 6 / (Pi * x) ^ 2], [1 .. infinity],
        ["Discrete", "PDF"]];
> Convert(X);
```

The converted Zipf random variable in the standard APPL *Dot* format is

$$\left[\left[x \rightarrow \frac{6}{(\pi x)^2} \right], [1 \dots \infty, 1, x \rightarrow x], ["Discrete", "PDF"] \right]$$

Example 6.12. Use the APPL **Convert** procedure to convert a Benford random variable (see Example 6.3) to the standard *NoDot* format.

The APPL statements below define X as a Benford random variable and convert it to its standard *NoDot* format.

```
> X := [[x -> log[10](1 + 1 / x)],
        [1, 2, 3, 4, 5, 6, 7, 8, 9],
        ["Discrete", "PDF"]];
> Convert(X);
```

The converted Benford random variable in the standard APPL *NoDot* format is

$$\left[\left[\frac{\ln(2)}{\ln(10)}, \frac{\ln(\frac{3}{2})}{\ln(10)}, \frac{\ln(\frac{4}{3})}{\ln(10)}, \frac{\ln(\frac{5}{4})}{\ln(10)}, \frac{\ln(\frac{6}{5})}{\ln(10)}, \frac{\ln(\frac{7}{6})}{\ln(10)}, \frac{\ln(\frac{8}{7})}{\ln(10)}, \frac{\ln(\frac{9}{8})}{\ln(10)}, \frac{\ln(\frac{10}{9})}{\ln(10)} \right], \right. \\ \left. [1, 2, 3, 4, 5, 6, 7, 8, 9], ["Discrete", "PDF"] \right]$$

As in the case of continuous random variables, there are several commonly-used random variables that are pre-defined and automated for input into APPL: *BenfordRV*, *BernoulliRV*, *BingoCoverRV*, *BinomialRV*, *BirthdayRV*, *GeometricRV*, *HypergeometricRV*, *NegativeBinomialRV*, *PoissonRV*, *UniformDiscreteRV*, and *ZipfRV*. This list of distributions, along with their parameters, is displayed every time APPL is entered. A binomial random variable X with $n = 3$ and $p = 1/2$, for example, can be defined by the APPL statement

```
> X := BinomialRV(3, 1 / 2);
```

which is much more compact than the standard list-of-sublists approach:

```
> X := [[1 / 8, 3 / 8, 3 / 8, 1 / 8], [0, 1, 2, 3],
        ["Discrete", "PDF"]];
```

6.2 Simple Algorithms

Now that the data structure for representing discrete random variables in APPL has been established, we present some elementary algorithms that can be applied to these random variables. Specifically, the algorithms described in this section conduct the following elementary operations:

- convert any of the six functional forms of a random variable (PDF, CDF, SF, HF, CHF, or IDF) into another functional form,
- plot any of the six functions for any fully-specified distribution, e.g., the CDF,
- determine common summary characteristics of a random variable, such as the mean and variance, and
- verify the validity of a particular PDF functional form.

Changing the Functional Form of a Random Variable. The functional form of a distribution can be converted to and from its six well-known forms, the PDF, CDF, SF, HF, CHF, and IDF. As in the case of continuous random variables, the procedures return functional forms when they are called with a single argument and scalars when an optional second argument is included. The names of the conversion procedures are PDF, CDF, SF, HF, CHF, and IDF, and the procedures can be applied to both discrete and continuous random variables.

The matrix in Table 6.2 shows how the PDF, CDF, SF, HF, and CHF distribution forms (given in the columns) can be found if one of the forms (given by the rows) is known. For example, if the CDF of a distribution is given, then its CHF can be determined for $x_i \in \Omega$ by

$$H(x_i) = -\log(S(x_i)) = -\log(1 - F(x_{i-1})),$$

Table 6.2. A 5×5 transition matrix for determining $f(x_i)$, $F(x_i)$, $S(x_i)$, $h(x_i)$, or $H(x_i)$ from any of the others for discrete distributions, where $F(x_0) \equiv 0$, $h(x_0) \equiv 0$, and when $|\Omega|$ is finite, $S(x_{|\Omega|+1}) \equiv 0$ and $H(x_{|\Omega|+1}) \equiv 0$.

	$f(x_i)$	$F(x_i)$	$S(x_i)$	$h(x_i)$	$H(x_i)$
$f(x)$	\bullet	$\sum_{j x_j \leq x_i} f(x_j)$	$\sum_{j x_j \geq x_i} f(x_j)$	$\frac{f(x_i)}{\sum_{j x_j \geq x_i} f(x_j)}$	$-\log \left(\sum_{j x_j \geq x_i} f(x_j) \right)$
$F(x)$	$F(x_i) - F(x_{i-1})$	\bullet	$1 - F(x_{i-1})$	$\frac{F(x_i) - F(x_{i-1})}{1 - F(x_{i-1})}$	$-\log \left(1 - F(x_{i-1}) \right)$
$S(x)$	$S(x_i) - S(x_{i+1})$	$1 - S(x_{i+1})$	\bullet	$\frac{S(x_i) - S(x_{i+1})}{S(x_i)}$	$-\log S(x_i)$
$h(x)$	$h(x_i) \prod_{j x_j < x_i} [1 - h(x_j)]$	$1 - \prod_{j x_j \leq x_i} [1 - h(x_j)]$	$\prod_{j x_j < x_i} [1 - h(x_j)]$	\bullet	$-\sum_{j x_j < x_i} \log(1 - h(x_j))$
$H(x)$	$e^{-H(x_i)} - e^{-H(x_{i+1})}$	$1 - e^{-H(x_{i+1})}$	$e^{-H(x_i)}$	$\frac{e^{-H(x_i)} - e^{-H(x_{i+1})}}{e^{-H(x_i)}}$	\bullet

where \log is the natural logarithm (\log base e) and $F(x_0) \equiv 0$. Some facts used to compute the entries in the matrix for $x_i \in \Omega$ are given in Leemis [50, pages 56–57, 73]:

- $h(x_i) = f(x_i)/S(x_i)$,
- $H(x_i) = -\log S(x_i)$,
- $S(x_i) = \prod_{j|x_j < x_i} (1 - h(x_j))$, and
- $H(x_i) = - \sum_{j|x_j < x_i} \log(1 - h(x_j))$.

An example of how to find one distribution form when another is known using the 5×5 matrix in Table 6.2 is given next.

Example 6.13. Given the hazard function

$$h(x) = 1/4 \quad x = 1, 2, \dots,$$

find $f(x_i)$ for $x_i = 1, 2, 3, \dots$.

Using the $(h(x), f(x_i))$ matrix element in Table 6.2,

$$f(x_i) = h(x_i) \prod_{j|x_j < x_i} [1 - h(x_j)] = \frac{1}{4} \prod_{j=1}^{x_i-1} \left[1 - \frac{1}{4}\right] = \frac{1}{4} \left(\frac{3}{4}\right)^{x_i-1}$$

for $x_i = 1, 2, 3, \dots$, which is a geometric distribution with $p = 1/4$. The geometric distribution is the only discrete distribution having both a constant hazard function and the memoryless property. This calculation is made in APPL with the statements

```
> X := [[x -> 1 / 4], [1 .. infinity], ["Discrete", "HF"]];
> PDF(X);
```

The algorithms for the PDF, CDF, SF, HF, and CHF procedures are similar and utilize the formulas in Table 6.2. When changing the “current form” of a random variable to a “new form,” the algorithm uses the Table 6.2 matrix element (“current form”, “new form”).

We now delve into some detail on how the PDF procedure works.

The PDF of the discrete random variable X is defined as $f(x) = \Pr(X = x)$ for all $x \in \Omega$. The `PDF(X, [x])` procedure returns either

1. the PDF of the random variable X in its APPL list-of-sublists format if only one argument is provided in the procedure call; i.e., `PDF(X)`, or
2. the probability value $\Pr(X = x)$ if both arguments are provided in the procedure call; i.e., `PDF(X, x)`.

Example 6.14. In order to illustrate many of the conversion operations, we use a discrete random variable X with PDF

$$f(x) = \frac{x}{6} \quad x = 1, 2, 3.$$

The formulas for its CDF, SF, HF, CHF, and IDF are:

$$\begin{aligned} F(x) &= \frac{x^2 + x}{12} & x = 1, 2, 3 \\ S(x) &= \frac{-x^2 + x + 12}{12} & x = 1, 2, 3 \end{aligned}$$

$$\begin{aligned}
h(x) &= \frac{2x}{-x^2 + x + 12} & x = 1, 2, 3 \\
H(x) &= -\log\left(\frac{-x^2 + x + 12}{12}\right) & x = 1, 2, 3 \\
F^{-1}(x) &= \left\lceil -\frac{1}{2} + \frac{1}{2}\sqrt{1 + 48x} \right\rceil & x = 1/6, 1/2, 1.
\end{aligned}$$

If we assign the variable **X** to the CDF of X using the *Dot* format, then the statement **PDF(X)** returns the PDF of X in it APPL list-of-sublists format:

```

> X := [[x -> (x ^ 2 + x) / 12], [1 .. 3],
        ["Discrete", "CDF"]];
> PDF(X);

```

yielding the PDF

$$\left[\left[x \rightarrow \frac{x}{6} \right], [1 .. 3], ["Discrete", "PDF"] \right]$$

Alternatively, the statements

```

> X := [[x -> (x ^ 2 + x) / 12], [1 .. 3],
        ["Discrete", "CDF"]];
> PDF(X, 2);

```

compute $\Pr(X = 2)$, which is $1/3$.

The algorithm in the PDF procedure first checks a discrete random variable X to determine if it has a *Dot* or *NoDot* support format. Although two distinct branches in the algorithm have been constructed to process random variables of each format separately, the basic functional relationships displayed in Table 6.2 are used in each branch. Whether X has a *Dot* or *NoDot* support format, one of the following bullets will be executed depending on X 's distribution form (and support format in the case of IDF). We assume $\Omega = \{x_1, x_2, \dots\}$, where $x_1 < x_2 < \dots$.

- If X has a PDF form, do nothing to X .
- If X has a CDF form, determine the PDF of X with the $(F(x), f(x_i))$ matrix element in Table 6.2:

$$f(x_i) = F(x_i) - F(x_{i-1}) \quad x_i \in \Omega,$$

where $i = 1, 2, \dots$ and $F(x_0) \equiv 0$.

- If X has a SF form, determine the PDF of X with the $(S(x), f(x_i))$ matrix element in Table 6.2:

$$f(x_i) = S(x_i) - S(x_{i+1}) \quad x_i \in \Omega,$$

where $i = 1, 2, \dots$ and $S(x_{|\Omega|+1}) \equiv 0$ when $|\Omega|$ is finite.

- If X has a HF form, determine the PDF of X with the $(h(x), f(x_i))$ matrix element in Table 6.2:

$$f(x_i) = h(x_i) \prod_{j|x_j < x_i} [1 - h(x_j)] \quad x_i \in \Omega,$$

where $i = 1, 2, \dots$ and $h(x_0) \equiv 0$.

- If X has a CHF form, determine the PDF of X with the $(H(x), f(x_i))$ matrix element in Table 6.2:

$$f(x_i) = e^{-H(x_i)} - e^{-H(x_{i+1})} \quad x_i \in \Omega,$$

where $i = 1, 2, \dots$ and $H(x_{|\Omega|+1}) \equiv 0$ when $|\Omega|$ is finite.

- If X has an IDF form with *Dot* support, then
 1. If there is a ceiling term as part of the inverse function, extract the expression under the ceiling in order to solve $F^{-1}(x) = y$. Referring back to X with PDF $f(x) = x/6$ for $x = 1, 2, 3$, the IDF of X (from Example 6.14) is

$$F^{-1}(x) = \left\lceil -\frac{1}{2} + \frac{1}{2}\sqrt{1+48x} \right\rceil,$$

for $x = 1/6, 1/2, 1$. Rewrite $F^{-1}(x)$ as

$$F^{-1}(x) = -\frac{1}{2} + \frac{1}{2}\sqrt{1+48x}.$$

This removal of the ceiling function can be done using the Maple `eval` procedure.

2. Solve the equation

$$F^{-1}(x) = y$$

for x , yielding $F(y)$. For our particular X , solving $-\frac{1}{2} + \frac{1}{2}\sqrt{1+48x} = y$ for x yields $F(x) = \frac{x^2+x}{12}$, replacing y with x .

3. Find the *appropriate* inverse. If there is more than one solution to the equation $F^{-1}(x) = y$, determine the correct inverse by testing which solution $F(x)$ correctly calculates $F(F^{-1}(s)) = s$ for s equal to one of the following. The choices for s are considered in the bulleted order.
 - if $\min\{\Omega\} = -\infty$ and $\max\{\Omega\} = \infty$, then $s = 1$.
 - if $\max\{\Omega\} \neq \infty$, then $s = \max\{\Omega\}$.
 - if $\min\{\Omega\} \neq -\infty$, then $s = \min\{\Omega\}$.
 - otherwise, $s = \frac{\min\{\Omega\} + \max\{\Omega\}}{2}$.

See the beginning of Section 4.2 for the continuous analog of these conditions.

4. Convert the CDF of X to its PDF form by the using the $(F(x), f(x_i))$ matrix element in Table 6.2.
 5. Determine the range of the PDF of X . The minimum support value is $F^{-1}(x_1)$ (where x_1 is the minimum support value of the IDF here) and the maximum support value is $F^{-1}(1)$. Since the minimum support value of the IDF for the example random variable X is $x_1 = 1/6$, then $F^{-1}(1/6) = 1$. Also, $F^{-1}(1) = 3$.
- If X has an IDF form and a *NoDot* support, then the second sublist of the APPL list-of-sublists contains the CDF values of X and the first sublist contains the CDF support values. To compute the PDF of X , the first and second sublists of the IDF are swapped to form the CDF, and then the PDF is computed using the $(F(x), f(x_i))$ matrix element in Table 6.2. The IDF of our example random variable X is

$$F^{-1}(x) = \begin{cases} 1 & x = \frac{1}{6} \\ 2 & x = \frac{1}{2} \\ 3 & x = 1. \end{cases}$$

The APPL list-of-sublists is

$$\left[[1, 2, 3], \left[\frac{1}{6}, \frac{1}{2}, 1 \right], [\text{“Discrete”, “IDF”}] \right]$$

The CDF is found by swapping the first two sublists:

$$\left[\left[\frac{1}{6}, \frac{1}{2}, 1 \right], [1, 2, 3], [\text{“Discrete”, “CDF”}] \right]$$

Finally, the PDF is

$$\begin{aligned} f(1) &= \frac{1}{6} \\ f(2) &= F(2) - F(1) = \frac{1}{3} \\ f(3) &= F(3) - F(2) = \frac{1}{2}. \end{aligned}$$

After the PDF has been computed, the algorithm checks whether one or two arguments were entered. If only one argument, X , is provided, i.e., $\text{PDF}(X)$, then the PDF of X is returned in its list-of-sublists format. Otherwise, if two arguments, X and x , are provided, then the probability value $\Pr(X = x)$ is computed. The algorithm distinguishes between the two support formats when determining this probability value:

- If X has the *Dot* support format, then
 1. If $x_i \in \Omega$, then compute and return the probability value $f(x_i)$ using the PDF formula already determined in the earlier part of the procedure.
 2. If the value $x_i \notin \Omega$, then return $\Pr(X = x_i) = 0$.

- If X has the *NoDot* support format, then
 1. Loop through the support values of the PDF searching for the value x_i in the second sublist. If x_i is found in position j , then its corresponding probability value is in position j of the first sublist. Return the j th element in the first sublist.
 2. If the value x_i is not found in the second sublist, then return $\Pr(X = x_i) = 0$.

Example 6.15. (Adapted from Hogg, et al. [37, page 44]) Cast a die two independent times and let X equal the absolute value of the difference of the two resulting values. The CDF of X is

$$F(x) = \begin{cases} \frac{1}{6} & x = 0 \\ \frac{11x - x^2 + 6}{36} & x = 1, 2, \dots, 5. \end{cases}$$

Determine the PDF of X .

Define the APPL *NoDot* format of $F(x)$ as the random variable \mathbf{X} . Then use the PDF procedure to determine X 's numeric PDF.

```
> X := [[1 / 6, 4 / 9, 2 / 3, 5 / 6, 17 / 18, 1],
        [0, 1, 2, 3, 4, 5], ["Discrete", "CDF"]];
> PDF(X);
```

The APPL *NoDot* format of $f(x)$ is

$$\left[\left[\frac{1}{6}, \frac{5}{18}, \frac{2}{9}, \frac{1}{6}, \frac{1}{9}, \frac{1}{18} \right], [0, 1, 2, 3, 4, 5], ["Discrete", "PDF"] \right]$$

which can be converted by hand to the formulaic PDF:

$$f(x) = \begin{cases} \frac{1}{6} & x = 0 \\ \frac{6 - x}{18} & x = 1, 2, \dots, 5. \end{cases}$$

Plotting Distributions. Especially for newcomers to probability theory, plots of the various functional forms of a random variable can provide insight into the distribution of that random variable. The `PlotDist` procedure requires only one argument, a random variable \mathbf{X} , but two additional optional arguments can be provided to indicate the plotting range, `low` and `high`. The procedure can plot a random variable in any of its six functional forms. The algorithm employed by the `PlotDist` procedure utilizes the list-of-sublists data structure to glean the necessary information for a plot. For discrete random variables, graphs for CDFs, SFs, CHFs, and IDFs, for instance, are plots of step functions with jumps at the support values specified in sublist two. Open and closed circles at the ends of the steps indicate exclusion and inclusion, respectively, of support values.

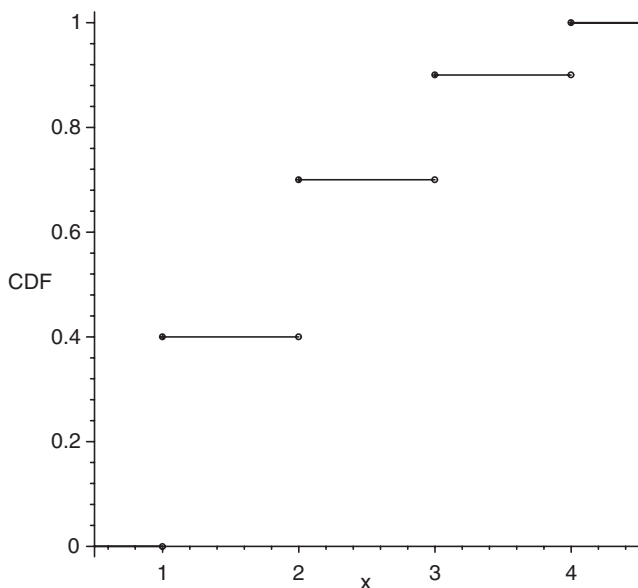


Fig. 6.1. CDF for $f(x) = (5 - x)/10$ for $x = 1, 2, 3, 4$.

Example 6.16. (Karian and Tanis [42, page 40]) Let

$$f(x) = (5 - x)/10 \quad x = 1, 2, 3, 4$$

be a PDF of the discrete random variable X . Use `PlotDist` to depict the CDF of X .

Instead of first converting X 's PDF form into a CDF form by hand, one can directly define the PDF and compose the CDF conversion with `PlotDist`. This leaves the conversion work to APPL. The plot of the desired CDF can be executed with the statements

```
> X := [[x -> (5 - x) / 10], [1 .. 4], ["Discrete", "PDF"]];
> PlotDist(CDF(X));
```

The corresponding plot is displayed in Figure 6.1.

Calculating Expected Values. If X is a discrete random variable with support $\Omega = \{x_1, x_2, \dots\}$ and PDF $f(x)$, then the mean (or expected value) of X is defined by

$$E[X] = \mu = \sum_{\text{all } x_i \in \Omega} x_i \cdot f(x_i).$$

More generally, if $g(X)$ is any function of X , such as $g(X) = X^2$, then

$$E[g(X)] = \sum_{\text{all } x_i \in \Omega} g(x_i) \cdot f(x_i).$$

The **ExpectedValue** procedure requires one argument, a random variable X , and can be given an additional argument, the function $g(X)$. When only one argument is provided for the **ExpectedValue** procedure, i.e., **ExpectedValue**(X), the mean of the random variable X is returned. When $g(x)$ is also given, the **ExpectedValue** procedure calculates the mean of $g(X)$. By default, the procedure assumes $g(X) = X$ when no second argument is given. The **Mean**, **Variance**, **Kurtosis**, and **Skewness** procedures all make use of the **ExpectedValue** procedure. This is efficient since variance, the coefficient of skewness, γ_1 , and the coefficient of kurtosis, γ_2 , for a random variable X can be calculated as

$$\begin{aligned}\text{Var}(X) &= E[(X - \mu)^2] = E[X^2] - \mu^2, \\ \gamma_1 &= E\left[\left(\frac{X - \mu}{\sigma}\right)^3\right] = \frac{E[X^3] - 3\mu E[X^2] + 2\mu^3}{[\text{Var}(X)]^{3/2}}, \text{ and} \\ \gamma_2 &= E\left[\left(\frac{X - \mu}{\sigma}\right)^4\right] = \frac{E[X^4] - 4\mu E[X^3] + 6\mu^2 E[X^2] - 3\mu^4}{[\text{Var}(X)]^2}.\end{aligned}$$

Example 6.17. (Ross [75, page 178]) Find the expected value and the variance of the number of times one must throw a die until the outcome one has occurred four times.

Since the random variable of interest is a negative binomial with parameters $r = 4$ and $p = 1/6$, the statements

```
> r := 4;
> p := 1 / 6;
> X := NegativeBinomialRV(r, p);
> ExpectedValue(X);
> Variance(X);
```

yield the correct results: $E[X] = 24$ and $\text{Var}(X) = 120$.

Example 6.18. (Ross [75, page 198]) Let X be a binomial random variable with parameters n and p . Show that

$$E\left[\frac{1}{X+1}\right] = \frac{1 - (1-p)^{n+1}}{(n+1)p}.$$

Since we want to compute $E\left[\frac{1}{X+1}\right]$, let $g(X) = 1/(X+1)$. The APPL statements

```
> unassign('p');
> X := BinomialRV(n, p);
> ExpectedValue(X, x -> 1 / (x + 1));
```

produce the desired expected value.

Verifying the Validity of a PDF. Let X be a discrete random variable defined on the support $\Omega = \{x_1, x_2, \dots\}$. A discrete probability PDF must satisfy the following two conditions:

$$\sum_{\text{all } x_i \in \Omega} \Pr(X = x_i) = 1 \quad \text{and} \quad \Pr(X = x_i) > 0 \text{ for each } x_i \in \Omega.$$

The APPL `VerifyPDF` procedure has one argument, a random variable \mathbf{X} , and the procedure's purpose is to verify that \mathbf{X} satisfies these two conditions. When procedure call `VerifyPDF(X)` is made, either a message is returned stating that \mathbf{X} has a valid PDF or that it does not. The following example illustrates the use of this procedure.

Example 6.19. (Karian and Tanis [42, page 62]) Verify that the probabilities of a geometric random variable with parameter p sum to 1.

The random variable X can be defined as a geometric random variable with parameter p using the pre-defined APPL random variable `GeometricRV(p)`. When the `GeometricRV` procedure receives a symbolic argument (parameter), it uses the Maple `assume` command to correctly assume that the symbolic parameter is inclusively between 0 and 1. Thus, the statements

```
> X := GeometricRV(p);
> VerifyPDF(X);
```

generate a message to the screen indicating that \mathbf{X} has a valid PDF.

This concludes our introduction of the data structure and some simple algorithms for discrete random variables. The next two chapters introduce nontrivial algorithms for two important operations on discrete random variables: finding the distribution of sums and finding the distribution of order statistics.

Sums of Independent Random Variables

This chapter presents an algorithm for computing the PDF of the sum of two independent discrete random variables, along with an implementation of the algorithm in APPL. Some examples illustrate the utility of this algorithm.

7.1 Preliminary Examples

An important operation in probability theory and statistical inference is to calculate the distribution of the sum of two independent random variables X and Y . Applications of sums of random variables appear in many areas of mathematics, probability theory, physics, and engineering. Most texts devote the majority of their attention to sums of continuous random variables, rather than discrete random variables. We focus here on the case in which X and Y are discrete random variables with integer-valued supports that are bounded below.

There are several approaches for computing the PDF of $Z = X + Y$ in the discrete case. The event $\{X + Y = z\}$, $z \in \mathbb{Z}$, can be written as the union of the disjoint events $\{X = \zeta, Y = z - \zeta\}$, $\{X = \zeta + 1, Y = z - (\zeta + 1)\}$, \dots , $\{X = z - \zeta, Y = \zeta\}$, where ζ is the minimum of the union of the support values of X and Y . The PDF $f_Z(z)$ of the convolution of the PDFs of the independent random variables X and Y (with integer supports) can be computed as

$$\begin{aligned} \Pr(Z = z) &= \Pr(X + Y = z) \\ &= \sum_{k=\zeta}^{z-\zeta} \Pr(X = k, Y = z - k) \\ &= \sum_{k=\zeta}^{z-\zeta} \Pr(X = k) \Pr(Y = z - k). \end{aligned}$$

The following example illustrates the use of this discrete convolution formula.

Example 7.1. (Ross [75, page 287]) If X and Y are independent Poisson random variables with respective parameters λ_1 and λ_2 , compute the PDF of $Z = X + Y$.

The PDF of a Poisson random variable X with parameter λ is

$$f_X(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad x = 0, 1, 2, \dots$$

Since $\zeta = \min\{0, 1, 2, \dots\} = 0$,

$$\begin{aligned} \Pr(Z = z) &= \Pr(X + Y = z) \\ &= \sum_{k=0}^z \Pr(X = k, Y = z - k) \\ &= \sum_{k=0}^z \Pr(X = k) \cdot \Pr(Y = z - k) \\ &= \sum_{k=0}^z \frac{e^{-\lambda_1} \lambda_1^k}{k!} \cdot \frac{e^{-\lambda_2} \lambda_2^{z-k}}{(z-k)!} \\ &= e^{-(\lambda_1 + \lambda_2)} \sum_{k=0}^z \frac{\lambda_1^k \lambda_2^{z-k}}{k! (z-k)!} \\ &= \frac{e^{-(\lambda_1 + \lambda_2)}}{z!} \sum_{k=0}^z \frac{z!}{k! (z-k)!} \lambda_1^k \lambda_2^{z-k} \\ &= \frac{(\lambda_1 + \lambda_2)^z e^{-(\lambda_1 + \lambda_2)}}{z!} \quad z = 0, 1, 2, \dots \end{aligned}$$

So $Z = X + Y$ has a Poisson distribution with parameter $\lambda_1 + \lambda_2$. As will be seen subsequently, the APPL procedure **Convolution** automates this process.

If one wants to sum more than two iid random variables, then the distribution function for Z can be determined by induction (Grinstead and Snell [31, page 286]). Let $Z_n = X_1 + X_2 + \dots + X_n$ be the sum of n independent random variables with common PDF $f_X(x)$ defined on the integers. Then the PDF of Z_1 is $f_X(x)$. We can write

$$Z_i = Z_{i-1} + X_i,$$

for $i = 2, 3, \dots, n$. Thus, since we know the distribution of X_i is $f_X(x)$, for $i = 1, 2, \dots, n$, we can find the distribution function of Z_n by induction. When summing more than two iid random variables, the APPL procedure **ConvolutionIID** uses this inductive process to compute the PDF of the sum.

For example, let X_1 , X_2 , and X_3 have PDFs $f_X(x) = 1/6$ for $x = 1, 2, \dots, 6$. If $Z_2 = X_1 + X_2$, then the PDF $f_{Z_2}(z)$ is

$$f_{Z_2}(z) = \begin{cases} \frac{z-1}{36} & z = 2, 3, \dots, 7 \\ \frac{13-z}{36} & z = 8, 9, \dots, 12. \end{cases}$$

The PDF of $Z_3 = X_1 + X_2 + X_3$ is the convolution of the PDFs of Z_2 and X_3 . For example, $\Pr(Z_3 = 4) = \Pr(Z_2 = 3) \cdot \Pr(X_3 = 1) + \Pr(Z_2 = 2) \cdot \Pr(X_3 = 2) = \frac{2}{36} \cdot \frac{1}{6} + \frac{1}{36} \cdot \frac{1}{6} = \frac{1}{72}$.

Due to the mathematical intractability in implementing the discrete convolution formula, $\sum_{k=\zeta}^{z-\zeta} \Pr(X = k, Y = z - k)$, for certain non-identically distributed random variables [e.g., $X \sim \text{Poisson}(\lambda)$, $Y \sim \text{geometric}(p)$] and the inefficiency in making computations with this formula for random variables with arbitrary supports (e.g., X with support $\{-216, -57, 23, 81\}$ and Y with support $\{-1002, -15, 2, 62, 211\}$), only certain convolutions can or should be computed using this formula. For random variables with arbitrary supports, the discrete convolution formula *can* be used, but it is often inefficient because one or both of the random variables have support values ranging over a large domain of non-adjacent integer values. The following example displays the inefficiency that can be encountered by using this formula, even for random variables with only a small number of support values.

Example 7.2. Suppose X and Y are independent discrete random variables with PDFs:

$$f_X(x) = \begin{cases} 0.15 & x = -3 \\ 0.25 & x = -1 \\ 0.1 & x = 2 \\ 0.3 & x = 6 \\ 0.2 & x = 8 \end{cases} \quad f_Y(y) = \begin{cases} 0.2 & y = -2 \\ 0.1 & y = 1 \\ 0.3 & y = 5 \\ 0.4 & y = 8. \end{cases}$$

Compute the PDF of $Z = X + Y$.

The support values for Z are

$$\{-5, -3, -2, 0, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 16\}.$$

We use the discrete convolution formula $\Pr(Z = z) = \sum_{k=\zeta}^{z-\zeta} \Pr(X = k, Y = z - k)$, where $\zeta = -3$, to compute $\Pr(Z = 4)$:

$$\begin{aligned} \Pr(Z = 4) &= \sum_{k=-3}^7 \Pr(X = k, Y = 4 - k) \\ &= \Pr(X = -3) \cdot \Pr(Y = 7) + \Pr(X = -2) \cdot \Pr(Y = 6) + \\ &\quad \Pr(X = -1) \cdot \Pr(Y = 5) + \Pr(X = 0) \cdot \Pr(Y = 4) + \\ &\quad \Pr(X = 1) \cdot \Pr(Y = 3) + \Pr(X = 2) \cdot \Pr(Y = 2) + \\ &\quad \Pr(X = 3) \cdot \Pr(Y = 1) + \Pr(X = 4) \cdot \Pr(Y = 0) + \\ &\quad \Pr(X = 5) \cdot \Pr(Y = -1) + \Pr(X = 6) \cdot \Pr(Y = -2) + \end{aligned}$$

$$\begin{aligned}
& \Pr(X = 7) \cdot \Pr(Y = -3) \\
&= 0.15 \cdot 0 + 0 \cdot 0 + 0.25 \cdot 0.3 + 0 \cdot 0 + 0 \cdot 0 + 0.1 \cdot 0 + \\
&\quad 0 \cdot 0.1 + 0 \cdot 0 + 0 \cdot 0 + 0.3 \cdot 0.2 + 0 \cdot 0 \\
&= 0 + 0 + 0.075 + 0 + 0 + 0 + 0 + 0 + 0 + 0.06 + 0 \\
&= 0.135.
\end{aligned}$$

The probabilities for the other support values of Z are computed in a similar fashion. Because of the tedious calculations needed to compute the PDF of Z by the discrete convolution formula, we compute it fully in the next example using moment generating functions (MGFs).

Unlike the discrete convolution formula, the moment generating function technique avoids computing sums that have zero probability of occurring, as illustrated below.

Example 7.3. Suppose X and Y are the discrete random variables defined in Example 7.2 and $Z = X + Y$. Find the PDF of Z using the moment generating function technique [37].

Since X and Y are independent, the MGF of Z is

$$\begin{aligned}
M_Z(t) &= E\left(e^{t(X+Y)}\right) \\
&= E\left(e^{tX} e^{tY}\right) \\
&= E\left(e^{tX}\right) E\left(e^{tY}\right) \\
&= M_X(t) M_Y(t).
\end{aligned}$$

The MGFs of X and Y , respectively, are

$$M_X(t) = E\left(e^{tX}\right) = 0.15e^{-3t} + 0.25e^{-t} + 0.1e^{2t} + 0.3e^{6t} + 0.2e^{8t}$$

and

$$M_Y(t) = E\left(e^{tY}\right) = 0.2e^{-2t} + 0.1e^t + 0.3e^{5t} + 0.4e^{8t}$$

for $-\infty < t < \infty$. Thus, the MGF of Z is

$$\begin{aligned}
M_Z(t) &= 0.03e^{-5t} + 0.05e^{-3t} + 0.015e^{-2t} + 0.045 + 0.045e^{2t} + \\
&\quad 0.01e^{3t} + 0.135e^{4t} + 0.06e^{5t} + 0.04e^{6t} + 0.16e^{7t} + 0.02e^{9t} + \\
&\quad 0.04e^{10t} + 0.09e^{11t} + 0.06e^{13t} + 0.12e^{14t} + 0.08e^{16t}
\end{aligned}$$

for $-\infty < t < \infty$. Then, the PDF of Z is found by matching the coefficients (mass values) with the associated coefficients in the exponents (support values):

$Z = z$	-5	-3	-2	0	2	3	4	5	6	7	9	10	11	13	14	16
$f_Z(z)$.03	.05	.015	.045	.045	.01	.135	.06	.04	.16	.02	.04	.09	.06	.12	.08

In complicated examples, especially those involving continuous random variables, using the moment generating function technique to obtain convolution functions can be more efficient than direct summation or integration. Along the same lines as the moment generating function technique, the probability generating function technique can be used for determining the PDF of the sum of discrete random variables with non-negative integer-valued supports. Unfortunately, implementing either of these techniques in a computer algebra system like Maple has drawbacks when the supports of the random variables X and/or Y are not integer-valued. These implementation issues are discussed in Section 7.4.

Besides the integration/summation and generating function methods already described, the characteristic functions of independent random variables can be used to compute convolutions (see Parzen [70, page 395]). In order to use this method, though, one must know the inversion formula of the PDF of a random variable in terms of its characteristic function. The complexity of the inversion makes this method unappealing for complicated or arbitrary distributions.

The purpose of this chapter is to present an algorithm for determining the distributions of sums of discrete random variables, especially those with finite arbitrary supports. Like the algorithm presented in Woodward [82], our algorithm handles well-known distributions, such as the binomial and Poisson. But, our algorithm was primarily written for arbitrary random variables whose supports have moderate cardinality. Computer algebra systems make it feasible to determine the distribution of a sum of these types of random variables.

Section 7.2 describes the algorithm for determining the PDF of a sum of two discrete random variables in conceptual terms. The algorithm that was constructed to compute the convolution appears in Section 7.3; implementation issues that arose when the algorithm was coded in APPL are given in Section 7.4. Section 7.5 provides a collection of examples that can be solved with the convolution algorithm.

7.2 Conceptual Algorithm Development

One way to compute the convolution of the PDFs of two independent discrete random variables is by what we call the “brute force method.” Let X have support x_1, x_2, \dots, x_n and Y have support y_1, y_2, \dots, y_m . This method does just what the name implies—it computes all possible sums between the support of X and the support of Y by brute force, e.g., $x_1 + y_1, x_1 + y_2, \dots, x_1 + y_m, x_2 + y_1, x_2 + y_2, \dots, x_n + y_{m-1}, x_n + y_m$. The sums are placed in a one-dimensional array, called s , of length $n \cdot m$. The corresponding probabilities for each of these sums, $f_X(x_1) \cdot f_Y(y_1), f_X(x_1) \cdot f_Y(y_2), \dots, f_X(x_n) \cdot f_Y(y_m)$, are stored in an one-dimensional array called *Probs*, also of length $n \cdot m$.

The probability in position $Probs_i$ corresponds to the sum in position s_i , $i = 1, 2, \dots, n \cdot m$.

For example, let X and Y be the random variables introduced in Example 7.2. The arrays s and $Probs$ for the random variables X and Y are

$$s = [-5, -2, 2, 5, -3, 0, 4, 7, 0, 3, 7, 10, 4, 7, 11, 14, 6, 9, 13, 16]$$

$$Probs = [0.03, 0.015, 0.045, 0.06, 0.05, 0.025, 0.075, 0.1, 0.02, 0.01, \\ 0.03, 0.04, 0.06, 0.03, 0.09, 0.12, 0.04, 0.02, 0.06, 0.08]$$

We assume that s is initially unsorted and may contain repeated values, such as 0, 4, and 7 in this particular example. The array s is then sorted and the elements in the array $Probs$ are rearranged in a corresponding way. After sorting, the arrays s and $Probs$ are

$$s = [-5, -3, -2, 0, 0, 2, 3, 4, 4, 5, 6, 7, 7, 7, 9, 10, 11, 13, 14, 16]$$

$$Probs = [0.03, 0.05, 0.015, 0.025, 0.02, 0.045, 0.01, 0.075, 0.06, 0.06, \\ 0.04, 0.1, 0.03, 0.03, 0.02, 0.04, 0.09, 0.06, 0.12, 0.08]$$

Finally, the redundancies in s are removed and the appropriate probabilities corresponding to those redundancies are combined in $Probs$. The final arrays are

$$s = [-5, -3, -2, 0, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 16]$$

$$Probs = [0.03, 0.05, 0.015, 0.045, 0.045, 0.01, 0.135, 0.06, 0.04, 0.16, \\ 0.02, 0.04, 0.09, 0.06, 0.12, 0.08]$$

One algorithm we employ to sort the array s is *insertion sort* [81]. When $n \cdot m$ is small, the simplicity of insertion sort makes it an appropriate choice. Unfortunately, for random variables X and Y with larger support sizes n and m , such as $n = m = 10$, insertion sort becomes inefficient. Since insertion sort is an $O(N^2)$ algorithm, where $N = n \cdot m$ in our setting, it is not an appropriate method for sorting lists containing more than one hundred or so elements. For this reason, another sorting algorithm, *heapsort* [81], is used to sort larger arrays. Heapsort uses a heap, which is a binary tree with special properties, to sort the array s . Heapsort is an $O(N \cdot \log(N))$ algorithm [11].

Heapsort builds the array s as a maximum heap data structure. It then swaps the maximum element (the root) of the heap with the element in the last array position $s_{n \cdot m}$. The heap is rebuilt with the remaining unsorted elements in array positions s_1 through $s_{n \cdot m - 1}$. The maximum element of the new heap is then swapped with the element in the second to last position of the array s , which is position $s_{n \cdot m - 1}$. Now the last two positions in s are sorted in ascending order. The heap structure is again restored with the remaining

unsorted elements in array positions s_1 through s_{n-m-2} . This swap and rebuild process repeats itself until all elements are removed from the unsorted region of the heap and placed in ascending order from the front to the back of the heap. Heapsort proved more efficient than insertion sort for large values of N .

Shellsort, an improved insertion sort, is the algorithm employed by the mathematical software package Maple to sort polynomials. Since Shellsort's "performance is quite acceptable in practice, even for N [the number of elements] in the tens of thousands" (Weiss [81, page 260]), we take advantage of Maple's sorting algorithm for polynomials (when possible) by using the moment generating function technique to compute the sum of discrete random variables. The MGFs for X and Y , denoted by $M_X(t)$ and $M_Y(t)$ respectively, are first computed. Next, the product of the MGFs, $M_Z(t)$, is computed. We manipulate the terms of the MGF with Maple's **expand** procedure so that they are written in a fashion that Maple interprets as polynomials terms. For example, if the MGF is $M_Z(t) = \frac{1}{3}e^{3t} + \frac{1}{6}e^{2t} + \frac{1}{2}e^{5t}$, then **expand**($M_Z(t)$) returns $M_Z(t)$ as $\frac{1}{3}(e^t)^3 + \frac{1}{6}(e^t)^2 + \frac{1}{2}(e^t)^5$. The terms of the resulting expanded MGF are then sorted in descending order by the constant appearing in the exponent of each e^t term. Sorting the example expression $M_Z(t)$ returns $\frac{1}{2}(e^t)^5 + \frac{1}{3}(e^t)^3 + \frac{1}{6}(e^t)^2$. The probability and support values are extracted from the terms of the expression $M_Z(t)$, and the PDF of the sum is formed. The PDF for the example expression $M_Z(t)$ is

$$f_Z(z) = \begin{cases} \frac{1}{6} & z = 2 \\ \frac{1}{3} & z = 3 \\ \frac{1}{2} & z = 5. \end{cases}$$

Although in theory this is an ideal method, Maple recognizes that expressions with non-integer valued constants in the exponents, such as $\frac{1}{3}e^{3.1t} + \frac{1}{6}e^{2.5t} + \frac{1}{2}e^{5.4t}$, are not truly polynomials and will incorrectly sort such expressions. Since the MGF $M_Z(t)$ may not always have integer constants in its exponents, the moment generating function technique for computing the PDF of a sum is only reasonable to use for integer supports. Using probability generating functions to compute the PDF of a sum of random variables results in the same complications. Further implementation issues faced by moment and probability generating functions are discussed in Section 7.4.

As suggested by Nicol [64], the sum array s can be constructed in such a way that the next largest sum element is placed in s as it is being built. Instead of constructing the array s first and then sorting it, our algorithm constructs s by sequentially appending the next ordered element. We refer to this method as the "moving heap method," and it involves building, deleting, and inserting sums into a minimum heap data structure. A minimum heap contains its smallest element in the root, rather than its largest as in a maximum heap data structure.

The idea behind this sorting algorithm is the construction of a two-dimensional "conceptual" array A , which has $m + 1$ rows and $n + 1$

columns. The array A is not instantiated to save on memory, but serves as a bookkeeping device to help explain the nature of the algorithm. As illustrated in Example 7.4, A is displayed in an unusual manner in order to resemble the axes in the Cartesian coordinate system. Without loss of generality, we assume that the supports of X and Y are arranged in increasing order; i.e., $x_1 < x_2 < \cdots < x_n$ and $y_1 < y_2 < \cdots < y_m$. The array cell (i, j) contains the sum $A_{i,j} = y_i + x_j$ for $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. The cells in row $m + 1$ of A hold a 0 or 1 for each column $j = 1, 2, \dots, n$ to indicate whether the cell in column j is “active,” which means its entry is in the heap H , consisting of the candidates for the next element to be appended to s . Thus $A_{m+1,j} = 0$ or 1 for $j = 1, 2, \dots, n$. Likewise, the cells in column $n + 1$ of A also hold a 0 or 1 for each row $i = 1, 2, \dots, m$ to indicate whether the cell in row i is “active,” so $A_{i,n+1} = 0$ or 1 for $i = 1, 2, \dots, m$. The cell $(m + 1, n + 1)$ is not used in the algorithm. Example 7.4 illustrates what is meant by an “active” cell. Occasionally, it will be convenient to have a notation for the matrix A with row $m + 1$ and column $n + 1$ deleted. This $m \times n$ matrix will be denoted by \hat{A} .

Since $x_j < x_{j+1}$ for $j = 1, 2, \dots, n-1$ and $y_i < y_{i+1}$ for $i = 1, 2, \dots, m-1$, the entry in cell (i, j) is always guaranteed to be less than both the entries in cells $(i + 1, j)$ and $(i, j + 1)$; i.e. $A_{i,j} < A_{i+1,j}$ and $A_{i,j} < A_{i,j+1}$. This result allows the algorithm to repeatedly move the smallest number of candidate entries for the next largest sum from the array A to the current heap H . Thus this algorithm moves from the southwest cell to the northeast cell of the array \hat{A} repeatedly placing the next largest sum into s after choosing it as the minimum element in the current heap H .

Theorem 7.1. Let $x_1 < x_2 < \cdots < x_n$ and $y_1 < y_2 < \cdots < y_m$ be finite real numbers. Let the $m \times n$ array \hat{A} be arranged as shown in Figure 7.1 and have entries $\hat{A}_{i,j} = y_i + x_j$, $i = 1, 2, \dots, m$ and $j = 1, 2, \dots, n$. Let the set C contain all (i, j) pairs such that $x_i + y_j \leq c$, where $c > y_1 + x_1$ is a fixed real number. Let P be the path from the northwest corner of \hat{A} to the southeast corner of \hat{A} that separates C from its complement C' . The smallest element in C' must occur immediately northeast of a point at which the path P changes direction from southward to eastward.

Proof. Suppose the smallest element in C' occurs in the cell (r, s) . Since $A_{r-1,s} < A_{r,s}$ and $A_{r,s-1} < A_{r,s}$, cells $(r - 1, s)$ and $(r, s - 1)$ are both in C . Thus the path P must pass between cell (r, s) and cells $(r, s - 1)$ and $(r - 1, s)$, changing direction from southward to eastward as it does. \square

Theorem 7.1 not only identifies the candidates for the next element to append to s , but it also shows how the list of candidates should be modified when its smallest element is removed. If $A_{p,q}$ is appended to s , then the path P corresponding to $c = A_{p,q}$ passes along the upper edge and right edge of cell (p, q) , implying that P may change direction just southwest of $A_{p+1,q}$ and $A_{p,q+1}$.

y_m	$y_m + x_1$	$y_m + x_2$	\dots	$y_m + x_n$
	\vdots	\vdots	\ddots	\vdots
y_2	$y_2 + x_1$	$y_2 + x_2$	\dots	$y_2 + x_n$
y_1	$y_1 + x_1$	$y_1 + x_2$	\dots	$y_1 + x_n$
	x_1	x_2		x_n

Fig. 7.1. Array \hat{A} where $x_1 < x_2 < \dots < x_n$ and $y_1 < y_2 < \dots < y_m$.

8	Ⓔ5	7	10	14	16
5	2	Ⓔ4	7	11	13
1	-2	0	3	7	9
-2	-5	-3	0	Ⓔ4	6
	-3	-1	2	6	8

Fig. 7.2. Array \hat{A} corresponding to Example 7.2. The thickened path P from the northwest corner to the southeast corner of \hat{A} delimits the set $C = \{(i, j) \mid y_i + x_j \leq 3\}$ from its complement C' . The circled entries lie just to the northeast of points where the path turns from southward to eastward. These three entries are candidates for the smallest entry in the cells of C' for $c = 3$.

As an example, Figure 7.2 displays the array \hat{A} associated with Example 7.2 with $c = 3$.

Since this process and its intricacies are best explained by an example, we'll reintroduce the discrete random variable X and Y which were first encountered in Example 7.2.

row 5		1	0	0	0	0
row 4	8					0
row 3	5					0
row 2	1					0
row 1	-2	-5				1
		-3	-1	2	6	8
		col 1	col 2	col 3	col 4	col 5
						col 6

Fig. 7.3. Array A with active cell $(1, 1)$, which contains the entry $A_{1,1} = -5$.

Example 7.4. Let X and Y have PDFs

$$f_X(x) = \begin{cases} 0.15 & x = -3 \\ 0.25 & x = -1 \\ 0.1 & x = 2 \\ 0.3 & x = 6 \\ 0.2 & x = 8 \end{cases} \quad f_Y(y) = \begin{cases} 0.2 & y = -2 \\ 0.1 & y = 1 \\ 0.3 & y = 5 \\ 0.4 & y = 8. \end{cases}$$

Use the “moving heap method” to determine the PDF of $Z = X + Y$.

Construct the 5×6 array A . Set $A_{i,n+1} = A_{i,6} = 0$ for $i = 1, 2, 3, 4$ and $A_{m+1,j} = A_{5,j} = 0$ for $j = 1, 2, 3, 4, 5$. The smallest value in A is positioned in cell $(1, 1)$ and is $A_{1,1} = y_1 + x_1 = -5$. The algorithm designates the cell $(1, 1)$ as an “active” cell in A by setting $A_{m+1,1} = A_{5,1} = 1$ and $A_{1,n+1} = A_{1,6} = 1$. The zeros in the other cells of row five and column six remain. Figure 7.3 displays this initial array. The entries of A increase in value as one moves up, to the right, or a combination of both (as in the Cartesian coordinate system).

As in the brute force method, let the one-dimensional array s of length $n \cdot m$ hold the sums of the supports of the random variables X and Y . The corresponding probabilities for each of these sums will again be stored in the one-dimensional array called *Probs*, also of length $n \cdot m$. Clearly, the first (smallest) sum to be placed in the first position of array s is $A_{1,1}$. Accordingly, $f_X(x_1) \cdot f_Y(y_1) = 0.03$ is placed in the *Probs* array in its first position, $Probs_1$. After setting $s_1 = A_{1,1} = -5$ and $Probs_1 = \Pr(Z = A_{1,1}) = f_X(x_1) \cdot f_Y(y_1) = 0.03$, the cell $(1, 1)$ becomes inactive. In order to reflect the absence of an active cell in the first row and first column, reset $A_{m+1,1} = A_{5,1} = 0$ and $A_{1,n+1} = A_{1,6} = 0$. The next two cells to become “active” (i.e., cells that may contain the next largest sum) in the array A are $(1, 2)$ and $(2, 1)$, with

row 5	1	1	0	0	0	
row 4 8						0
row 3 5						0
row 2 1	-2					1
row 1 -2	-5	-3				1
	-3	-1	2	6	8	
	col 1	col 2	col 3	col 4	col 5	col 6

Fig. 7.4. Array A after $A_{1,1}$ has been processed and added to the one-dimensional sum array s . The cells $(1, 2)$ and $(2, 1)$ are active, as indicated by the ones in cells $(1, 6)$, $(5, 2)$, $(2, 6)$, and $(5, 1)$.

$A_{1,2} = y_1 + x_2 = -3$ and $A_{2,1} = y_2 + x_1 = -2$. Since cell $(1, 2)$ in A is now active, reset $A_{1,6} = 1$ and set $A_{5,2} = 1$. Similarly, since cell $(2, 1)$ is active, set $A_{2,6} = 1$ and reset $A_{5,1} = 1$. The purpose of these ones and zeros along the boundary of the A array is to assure that there is no more than one active cell in each row and column. Figure 7.4 contains the current state of array A .

The values $A_{1,2}$ and $A_{2,1}$ now form the minimum heap H , resulting in $H = \{A_{1,2}, A_{2,1}\} = \{-3, -2\}$. Informally, a *heap* is a complete binary tree with a special ordering property of its nodes. A complete binary tree is a tree that is completely filled with the possible exception of the bottom level, which is filled from left to right. Figure 7.5 contains illustrations of structures which are and are not complete binary trees. Each node of the tree has one parent, except the *root* of the tree, which has no parent. In a *minimum heap*, the smallest element of the heap is contained in its root. In the upper-right tree in Figure 7.5, a is the root of the tree. Nodes b and c are a 's *children*, where b is the *left child* and c is the *right child*. (According to the definition of a complete binary tree, when a node above the bottom level of the tree has only one child, it must be a left child.) Node b is the parent to nodes d and e . The *height* of a tree is the number of nodes from the root to a node at the bottom level of the tree. For example, the heights of the top trees in Figure 7.5 are three, three, and four, respectively.

Thus, a minimum heap is a complete binary tree with the special ordering property that each parent node contains a value less than or equal to the values in its children's nodes. Because of this ordering property, the smallest value in a minimum heap will always be at the root.

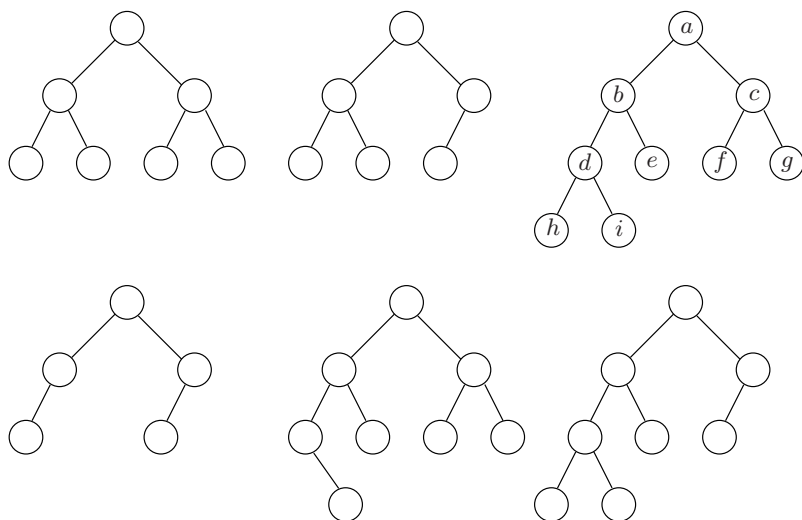


Fig. 7.5. Six binary trees. The top three trees are complete binary trees and the bottom three are not.

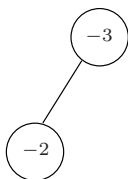


Fig. 7.6. Heap H containing entries $A_{1,2} = -3$ and $A_{2,1} = -2$.

The binary heap H formed with the values $A_{1,2}$ and $A_{2,1}$ is in Figure 7.6. The next sum to be entered into s in position s_2 is the root of the heap, i.e., the smallest element in H . Since $A_{1,2} = -3$ is the root, it is removed from the heap H and placed in s_2 , while its corresponding probability is placed in $Probs_2$. Because the entry $A_{1,2}$ is removed from the array A , reset $A_{1,6} = 0$ and $A_{5,2} = 0$ to indicate that row one and column two no longer contain an active cell. After these changes, array A is displayed in Figure 7.7.

After setting cell $(1, 2)$ to inactive, the two cells that are candidates to become active (if the corresponding row and column do not already contain an active cell) are cells $(3, 1)$ and $(2, 2)$. Since row two contains the active cell $(2, 1)$, and by design $A_{2,1} < A_{2,2}$, entry $A_{2,2}$ is not activated. However, cell $(1, 3)$ does become active, and its entry is $A_{1,3} = y_1 + x_3 = 0$. Hence, $A_{1,6} = 1$ and $A_{5,3} = 1$. After these changes, array A is displayed in Figure 7.8.

row 5		1	0	0	0	0
row 4	8					0
row 3	5					0
row 2	1	-2				1
row 1	-2	-5	-3			0
		-3	-1	2	6	8
		col 1	col 2	col 3	col 4	col 5

Fig. 7.7. Array A after $A_{1,2} = -3$ is processed and appended to s . Cell (2, 1) is the only active cell. Candidates to become active are cells (1, 3) and (2, 2). Cell (2, 2) cannot become active since row two already contains an active cell.

row 5		1	0	1	0	0
row 4	8					0
row 3	5					0
row 2	1	-2				1
row 1	-2	-5	-3	0		1
		-3	-1	2	6	8
		col 1	col 2	col 3	col 4	col 5

Fig. 7.8. Array A with active cells (1, 3) and (2, 1).

The entry $A_{1,3}$ is inserted into the heap H , resulting in $H = \{A_{2,1}, A_{1,3}\} = \{-2, 0\}$. After the addition of $A_{1,3}$, the heap H is displayed in Figure 7.9. The minimum element, $A_{2,1}$, is removed from the root of the heap and placed in the sum array s in position s_3 . Its corresponding probability is placed in $Probs_3$.

The two cells that may enter the array A after the removal of the $A_{2,1}$ entry are in cells (2, 2) and (3, 1), as indicated by the arrows in Figure 7.10. Both cells (2, 2) and (3, 1) become active, and their values are $A_{2,2} = y_2 + x_2 = 0$ and $A_{3,1} = y_3 + x_1 = 2$. Hence, $A_{2,6} = 1$, $A_{5,2} = 1$, $A_{3,6} = 1$, and $A_{5,1} = 1$, as displayed in Figure 7.11. Entries

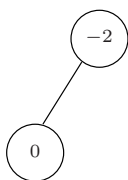


Fig. 7.9. Heap H containing entries $A_{2,2} = -2$ and $A_{1,3} = 0$.

row 5		0	0	1	0	0	
row 4	8						0
row 3	5						0
row 2	1	-2					0
row 1	-2	-5	-3	0			1
		-3	-1	2	6	8	
		col 1	col 2	col 3	col 4	col 5	col 6

Fig. 7.10. Array A after $A_{2,2} = -2$ is removed. Cell $(3, 1)$ is the only active cell. Candidates to become active are cells $(2, 2)$ and $(3, 1)$.

row 5		1	1	1	0	0	
row 4	8						0
row 3	5	2					1
row 2	1	-2	0				1
row 1	-2	-5	-3	0			1
		-3	-1	2	6	8	
		col 1	col 2	col 3	col 4	col 5	col 6

Fig. 7.11. Array A with active cells $(1, 3)$, $(2, 2)$, and $(3, 3)$.

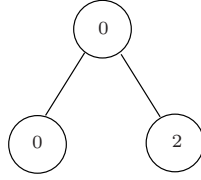


Fig. 7.12. Heap H with entries $A_{1,3} = 0$, $A_{2,2} = 0$ and $A_{1,3} = 2$.

row 5		0	0	0	1	0	
row 4	8	5	7	10			0
row 3	5	2	4	7	11		1
row 2	1	-2	0	3	7	9	0
row 1	-2	-5	-3	0	4	6	0
		-3	-1	2	6	8	
		col 1	col 2	col 3	col 4	col 5	col 6

Fig. 7.13. Array A with its seventeenth active cell $(3, 4)$.

$A_{2,2}$ and $A_{3,1}$ are inserted into the heap H , and H is again rebuilt as $H = \{A_{1,3}, A_{2,2}, A_{3,1}\} = \{0, 0, 2\}$. Its structure is displayed in Figure 7.12.

Moving ahead to the seventeenth pass through the construction of the array A , its appearance is displayed in Figure 7.13. $A_{3,4} = 11$ is placed in s_{17} , and values $A_{3,5} = 13$ and $A_{4,4} = 14$ are activated in the array A and inserted into the heap H . Since $A_{3,5}$ is the root of the heap, it is deleted and placed in s_{18} . No new element is allowed to enter H , so the root element of the heap is now $A_{4,4} = 14$, and it is removed and placed in s_{19} . The last entry to be activated is $A_{4,5} = 16$, and it is placed in position s_{20} of the sum array s .

Thus, after twenty iterations of this process, s and $Probs$ arrays are

$$s = [-5, -3, -2, 0, 0, 2, 3, 4, 4, 5, 6, 7, 7, 7, 9, 10, 11, 13, 14, 16]$$

$$Probs = [0.03, 0.05, 0.015, 0.025, 0.02, 0.045, 0.01, 0.075, 0.06, 0.06, \\ 0.04, 0.1, 0.03, 0.03, 0.02, 0.04, 0.09, 0.06, 0.12, 0.08]$$

which are the same arrays encountered by using the moment generating function technique. The redundancies are removed from s and

the appropriate probabilities are combined in *Probs* to complete the algorithm. (This could have been embedded into the iterative steps of the algorithm to save memory.) Thus, the PDF of $Z = X + Y$ is the same as determined in Example 7.3.

7.3 Algorithm

The algorithm for the procedure `Convolution(X, Y)` returns the convolution of the PDFs of the random variables X and Y . A brief description of the algorithm for discrete random variables follows.

If X and Y are discrete, their supports, finite or infinite, dictate which of the methods described in Section 7.2 is used to compute the convolution. The convolution of the PDFs of X and Y with finite support is computed either using the `BruteForceMethod` or `MovingHeapMethod` procedures, whose algorithms appear in the appendices of Evans and Leemis [23]. The PDF of the sum of Z is stored and returned in the usual list-of-sublists format. The list of elements $f(z_1), f(z_2), \dots, f(z_{n \cdot m})$ are the probability values of Z , while $z_1, z_2, \dots, z_{n \cdot m}$ are its support values. The one-dimensional array s is created to contain the unique sums extracted from the array A . Similarly, the one-dimensional array *Probs* is created to hold the probability values corresponding to the sums in s .

The zeros (if any) at the end of the *Probs* array do not represent probability values; they correspond to the zeros in the *Probs* array that are not support values. These extra zeros indicate that there are redundant values in the support of Z . The non-zero probability values are removed from *Probs* and placed in the array *FinalProbs*. The support values that correspond to the removed probability values are removed and placed in the array *FinalSupport*.

If the supports of the random variables X and Y are infinite, either the discrete convolution formula is used to compute the convolution or the APPL procedure `MGF` is used to determine the product of the MGFs of X and Y .

If either X or Y has infinite support, while the other has finite support, the product of their MGFs is returned. If both X and Y have infinite support and the discrete convolution formula formed with their PDFs results in an intractable sum, then the product of their MGFs is returned. Otherwise, the discrete convolution formula is used to determine the convolution of their PDFs.

Unless the MGF of Z or the PDF of the sum of X and Y (with $n = 1$ and $m = 1$) has already been returned, the PDF $f_Z(z)$ is returned.

Procedure `Convolution`: Computes the PDF of the sum two independent random variables (indentation indicates nesting).

Input: The random variables X and Y in the usual APPL list-of-sublists format. Assume the random variables X and Y are input in their PDF forms. The support of X is Ω_X and the support of Y is Ω_Y .

Output: The PDF of $Z = X + Y$.

Algorithm:

```

if  $X$  and  $Y$  are continuous
     $f_Z(z) \leftarrow \int_{-\infty}^{\infty} f_X(z-y) f_Y(y) dy$ 
else if  $X$  and  $Y$  are discrete
    if  $X$  and  $Y$  have finite support
         $n \leftarrow |\Omega_X|$ 
         $m \leftarrow |\Omega_Y|$ 
        if  $n \cdot m \leq 100$ 
             $Z \leftarrow \text{BruteForceMethod}(X, Y)$ 
        else
             $Z \leftarrow \text{MovingHeapMethod}(X, Y)$ 
    if ( $n = 1$  and  $m = 1$ )
        return( $f_Z(z)$ )
    dimension  $s[n \cdot m]$ 
    dimension  $Probs[n \cdot m]$ 
    for  $i \leftarrow 1$  to  $n \cdot m$ 
         $s_i \leftarrow 0$ 
         $Probs_i \leftarrow 0$ 
     $s_1 \leftarrow z_1$ 
     $s_2 \leftarrow z_2$ 
     $Probs_1 \leftarrow f(z_1)$ 
     $k \leftarrow 2$ 
     $j \leftarrow 2$ 
    while ( $k < n \cdot m$ ) do
         $Probs_j \leftarrow Probs_j + f(z_k)$ 
        if  $z_k \neq z_{k+1}$  then
             $j \leftarrow j + 1$ 
             $s_j \leftarrow z_{k+1}$ 
             $k \leftarrow k + 1$ 
         $Probs_j \leftarrow Probs_j + f(z_k)$ 
     $NumZeros \leftarrow 0$ 
    for  $i \leftarrow n \cdot m$  to 1 by  $-1$  while  $Probs_i = 0$ 
         $NumZeros \leftarrow NumZeros + 1$ 
    dimension  $FinalProbs[1, n \cdot m - NumZeros]$ 
    dimension  $FinalSupport[1, n \cdot m - NumZeros]$ 
    for  $i \leftarrow 1$  to  $(n \cdot m - NumZeros)$ 
         $FinalProbs_i \leftarrow Probs_i$ 
         $FinalSupport_i \leftarrow s_i$ 
     $f_Z(z) \leftarrow [FinalSupport, FinalProbs]$ 
    else if ( $X$  or  $Y$  has infinite support or  $X$  and  $Y$  have infinite support
        with intractable discrete convolution sum)
         $mgfx \leftarrow \text{MGF}(X)$ 
         $mgfy \leftarrow \text{MGF}(Y)$ 

```

```

    mgfprod ← mgfx · mgfy
    return(mgfprod)
else
    discrete convolution formula
    fZ(z) ←  $\sum_{k=0}^z (f_X(z)) \cdot (f_Y(z - k))$ 
else
    print("ERROR: X and Y must both be continuous or discrete")
    return
return(fZ(z))

```

7.4 Implementation Issues

The algorithm for `Convolution` has been implemented and included in APPL's suite of procedures. One portion of the algorithm for the `Convolution` procedure involves computing every possible sum between the support of the random variables X and Y . At the same time, the probabilities for those sums are calculated. Earlier we have referred to this as the "brute force method." The list of sums are ordered and redundancies combined. The corresponding probability values are repositioned to match their corresponding sums. One important reason for sorting the sums is that all other APPL procedures assume that discrete distributions, written in their list-of-sublists form, have supports listed in increasing order without repeated values. To be consistent with the APPL language and textbooks, the sums are sorted. Also, placing the values of the support into a list in sorted order means that tied $y_i + x_j$ values *can* be combined dynamically as the algorithm proceeds.

The first sorting method used to sort the list of sums was insertion sort. It was chosen because of its straightforward code and efficiency in computing convolutions when the support sizes of the random variables are small. Unfortunately, as the supports of random variables grow larger [e.g., random variables with 50 random numbers chosen on the interval $(-1, 1)$ as a support], the time used to compute the PDF of the sum becomes unreasonably long. A faster sorting method for larger list sizes is heapsort. Heapsort is employed in the `Convolution` procedure for sorting the list of sums created by the "brute force method."

Maple uses Shellsort to sort polynomials. In order to use the Shellsort procedure in Maple, the MGFs of X and Y need to be computed. The product of the MGFs of X and Y is an expression composed of exponential terms e^{kt} , where $k \in \mathbb{R}$, $t > 0$. Letting $u = e^t$, the MGF of the product can be rewritten as a polynomial-type expression. For example, if $u = e^t$, then $M_Z(t) = \frac{1}{3}e^{3t} + \frac{1}{6}e^{2t} + \frac{1}{2}e^{5t}$ for $-\infty < t < \infty$ can be rewritten $M_Z(t) = \frac{1}{3}u^3 + \frac{1}{6}u^2 + \frac{1}{2}u^5$ for $u > 0$. The Shellsort procedure sorts the polynomial-like expression, and the PDF of the convolution of X and Y is retrieved from this expression. Instead of MGFs, probability generating functions (PGFs) can be used in the process. MGFs were chosen over PGFs since PGFs can only be formed when the discrete distribution has non-negative integer support.

The method of computing convolutions via their MGFs was abandoned after realizing that Maple can only sort “true” polynomial expressions in which variables are raised to non-negative integer powers. Thus using MGFs would be too restrictive for a Maple implementation. Maple is unable to sort an expression that has non-integers in the powers of the polynomial terms. For example, Maple cannot sort the expression $\frac{1}{3}(e^t)^{3/2} + \frac{1}{6}(e^t)^{1/2} + \frac{1}{2}(e^t)^{5.5}$. Since the `Convolution` procedure was intended to be used on arbitrary distributions, which could indeed have negative, non-integer supports, the MGF method was removed from the procedure `Convolution`. The extra time involved in checking for appropriate values also affected the algorithm’s efficiency.

Unfortunately, for random variables with large finite support sizes, heap-sort was also inefficient. Nicol [64] suggested constructing a heap dynamically and sorting the list of sums sequentially, instead of waiting until the list of sums is built and then sorting it. The heap will always contain $\min\{m, n\}$ or fewer entries. After constructing the algorithm for this method, which is called `MovingHeapMethod`, the `Convolution` procedure was tested on random variables with large supports by using the `BruteForceMethod` with insertion sort, the `BruteForceMethod` with heapsort, and the `MovingHeapMethod`. A brief comparison analysis of the three methods suggested that `MovingHeapMethod` yielded the best times for computing sums of random variables with large supports. Test cases of random variables with increasing support sizes were performed to confirm this claim.

The test involved generating random numbers between -1 and 1 and making them the support values for the random variables X and Y . The random numbers were sorted and placed into the second sublist in the list-of-sublists format for the random variables that conforms to the system of the APPL language. The probabilities, which had no effect on the efficiency of the different algorithms, were assigned to be equal for all support values; i.e., if X ’s support consisted of 50 values, then each support value’s probability was $1/50$. The times for determining the PDF of the sum of random variables of increasing support sizes appears in [23].

When either one or both random variables’ supports are infinite, either the convolution of their PDFs is computed via the discrete convolution formula or the product of their MGFs is determined. If one of the random variables has infinite support, while the other has finite support, the product of their MGFs is returned. At this time, APPL does not contain a procedure to convert the MGF of a random variable to its PDF form.

Having two random variables with infinite support does not guarantee that the PDF of their sum can be determined by the discrete convolution formula. Only tractable summations, such as the convolution formula for two Poisson random variables as in Example 7.1, can be computed. This means that instead of determining the PDF for the convolution of the PDFs of some random variables, such as a Poisson with parameter $\lambda = 5$ and a geometric with

parameter $p = 0.3$, the **Convolution** procedure only computes the product of their MGFs.

7.5 Examples

The following examples use the algorithm described in Section 7.3 to determine the PDF of the sum of independent random variables. Examples for a variety of random variables are provided to illustrate the utility of the algorithm. Returning first to Examples 7.1 and 7.2 introduced in Section 7.1, we can use the **Convolution** procedure to determine the PDF of $Z = X + Y$.

Example 7.5. If X and Y are independent Poisson random variables with respective parameters λ_1 and λ_2 , compute the PDF of $Z = X + Y$.

In APPL, define X as a Poisson random variable with parameter `lambda1` and Y as a Poisson random variable with parameter `lambda2`. The Poisson random variable is also predefined in APPL. The PDF of $Z = X + Y$ is found with the statements

```
> X := PoissonRV(lambda1);
> Y := PoissonRV(lambda2);
> Z := Convolution(X, Y);
```

which returns the PDF of Z as

$$\left[\left[z \rightarrow \frac{(z+1) e^{-\lambda_1 - \lambda_2} (\lambda_2 + \lambda_1)^z}{(z+1)!} \right], [0 \dots \infty], [\text{"Discrete"}, \text{"PDF"}] \right]$$

Using the Maple **simplify** procedure, the resulting PDF after simplification is

$$f(z) = \frac{(\lambda_2 + \lambda_1)^z e^{-\lambda_1 - \lambda_2}}{\Gamma(z+1)} \quad z = 0, 1, \dots,$$

which is easy to recognize in its standard form as

$$f(z) = \frac{(\lambda_1 + \lambda_2)^z e^{-\lambda_1 - \lambda_2}}{z!} \quad z = 0, 1, \dots$$

We are fortunate in this example that Maple can compute the PDF by the discrete convolution formula by simplifying the sum $\sum_{k=0}^x \frac{\lambda_1^k e^{-\lambda_1} \lambda_2^{x-k} e^{-\lambda_2}}{k! (x-k)!}$. Unfortunately, Maple can only simplify certain expressions, so in some instances we cannot compute the PDF by the discrete convolution formula.

Example 7.6. X and Y are independent discrete random variables with PDFs defined as

$$f_X(x) = \begin{cases} 0.15 & x = -3 \\ 0.25 & x = -1 \\ 0.1 & x = 2 \\ 0.3 & x = 6 \\ 0.2 & x = 8 \end{cases} \quad f_Y(y) = \begin{cases} 0.2 & y = -2 \\ 0.1 & y = 1 \\ 0.3 & y = 5 \\ 0.4 & y = 8. \end{cases}$$

Find the PDF of Z .

Define the random variables X and Y in APPL's list-of-sublists format. Compute the PDF of $Z = X + Y$ with the statements

```
> X := [[0.15, 0.25, 0.1, 0.3, 0.2], [-3, -1, 2, 6, 8],
        ["Discrete", "PDF"]];
> Y := [[0.2, 0.1, 0.3, 0.4], [-2, 1, 5, 8],
        ["Discrete", "PDF"]];
> Z := Convolution(X, Y);
```

which return the PDF of Z as

```
[[0.03, 0.05, 0.015, 0.045, 0.045, 0.01, 0.135, 0.06, 0.04, 0.16, 0.02, 0.04,
    0.09, 0.06, 0.12, 0.08],
 [-5, -3, -2, 0, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 16], ["Discrete", "PDF"]]
```

Example 7.7. Let X and Y be independent random variables; X assumes three possible values 0, 1, 3 with probabilities $1/2$, $3/8$, and $1/8$, and Y assumes two possible values 0 and 1 with probabilities $1/3$ and $2/3$. Find the PDF of the random variable $Z = X + Y$ [10, page 136].

By hand, we can compute the PDF of Z with probability generating functions. The PGFs $G_X(t)$ and $G_Y(t)$ of X and Y , respectively, are

$$G_X(t) = \frac{1}{8}t^3 + \frac{3}{8}t + \frac{1}{2} \quad -\infty < t < \infty,$$

$$G_Y(t) = \frac{2}{3}t + \frac{1}{3} \quad -\infty < t < \infty.$$

Thus, the PGF of $Z = X + Y$ is the product of $G_X(t)$ and $G_Y(t)$:

$$G_Z(t) = \frac{1}{12}t^4 + \frac{1}{24}t^3 + \frac{1}{4}t^2 + \frac{11}{24}t + \frac{1}{6} \quad -\infty < t < \infty,$$

and hence the PDF of Z is

$$f_Z(z) = \begin{cases} \frac{1}{6} & z = 0 \\ \frac{11}{24} & z = 1 \\ \frac{1}{4} & z = 2 \\ \frac{1}{24} & z = 3 \\ \frac{1}{12} & z = 4. \end{cases}$$

In APPL, defining **X** and **Y** as lists-of-sublists and then applying the **Convolution** procedure achieves the same result.

```
> X := [[1 / 2, 3 / 8, 1 / 8], [0, 1, 3],
         ["Discrete", "PDF"]];
> Y := [[1 / 3, 2 / 3], [0, 1], ["Discrete", "PDF"]];
> Z := Convolution(X, Y);
```

Other measures of a distribution, such as the mean and variance, can be found with the use of additional APPL procedures, as illustrated in the next example.

Example 7.8. Let X_1 and X_2 be observations of a random sample of size $n = 2$ drawn from a distribution with PDF $f(x) = \frac{x}{6}$, $x = 1, 2, 3$. Find the PDF of $Y = X_1 + X_2$, and determine the mean and variance of the sum [38, page 297].

The PGF of X_1 and X_2 is

$$G_{X_1}(t) = G_{X_2}(t) = \frac{1}{2}t^3 + \frac{1}{3}t^2 + \frac{1}{6}t \quad -\infty < t < \infty.$$

Thus, $G_Y(t)$ is

$$G_Y(t) = \frac{1}{4}t^6 + \frac{1}{3}t^5 + \frac{5}{18}t^4 + \frac{1}{9}t^3 + \frac{1}{36}t^2 \quad -\infty < t < \infty,$$

and $f_Y(y)$ is

$$f_Y(y) = \begin{cases} \frac{1}{36} & y = 2 \\ \frac{1}{9} & y = 3 \\ \frac{5}{18} & y = 4 \\ \frac{1}{3} & y = 5 \\ \frac{1}{4} & y = 6. \end{cases}$$

The mean and the variance of Y , respectively, are

$$E[Y] = G'_Y(t) \Big|_{t=1} = \left[\frac{3}{2}t^5 + \frac{5}{3}t^4 + \frac{10}{9}t^3 + \frac{1}{3}t^2 + \frac{1}{18}t \right]_{t=1} = \frac{14}{3}$$

and

$$G''_Y(1) + G'_Y(1) - [G'_Y(1)]^2 = \frac{10}{9}.$$

In APPL, the mean and variance of Y are computed with the statements

```
> X := [[x -> x / 6], [1 .. 3], ["Discrete", "PDF"]];
> Y := Convolution(X, X);
```

```
> Mean(Y);
> Variance(Y);
```

which return the mean and variance computed by hand.

Example 7.9. Find the probability of obtaining a total of 14 in a single toss of four dice [36, page 230].

Let X be the PDF $f_X(x) = \frac{1}{6}$, $x = 1, 2, \dots, 6$. The PGF G of X is

$$G_X(t) = \frac{1}{6}t^6 + \frac{1}{6}t^5 + \frac{1}{6}t^4 + \frac{1}{6}t^3 + \frac{1}{6}t^2 + \frac{1}{6}t \quad -\infty < t < \infty.$$

The PDF of $Z = X_1 + X_2 + X_3 + X_4$ can be found by computing $[G_X(t)]^4$, which is

$$\begin{aligned} [G_X(t)]^4 = & \frac{1}{1296}t^{24} + \frac{1}{324}t^{23} + \frac{5}{648}t^{22} + \frac{5}{324}t^{21} + \frac{35}{1296}t^{20} + \\ & \frac{7}{162}t^{19} + \frac{5}{81}t^{18} + \frac{13}{162}t^{17} + \frac{125}{1296}t^{16} + \frac{35}{324}t^{15} + \\ & \frac{73}{648}t^{14} + \frac{35}{324}t^{13} + \frac{125}{1296}t^{12} + \frac{13}{162}t^{11} + \frac{5}{81}t^{10} + \\ & \frac{7}{162}t^9 + \frac{35}{1296}t^8 + \frac{5}{324}t^7 + \frac{5}{648}t^6 + \frac{1}{324}t^5 + \frac{1}{1296}t^4 \end{aligned}$$

for $-\infty < t < \infty$. Thus, $\Pr(Z = 14) = 73/648$. In APPL, define \mathbf{X} as a uniform discrete random variable (predefined in APPL) with parameter $1/6$. The procedure `ConvolutionIID(X, n)` computes the PDF of the convolution of n iid random variables \mathbf{X} . This procedure contains a Maple `for` loop which calls `Convolution` n times.

```
> X := UniformDiscreteRV(1, 6);
> Z := ConvolutionIID(X, 4);
> PDF(Z, 14);
```

The APPL procedure `PDF` computes the probability that Z is 14 correctly as $73/648$.

Examples 7.10 through 7.12 are from the article “Getting Normal Probability Approximations without Using Normal Tables” by Thompson [79]. His article discusses an alternate approach to approximating probabilities involving sums of discrete random variables using the PDF for the normal distribution. He lets S denote the sum of n independent discrete random variables, and assumes that S can take on consecutive integer values. Letting $\mu = E(S)$ and $\sigma^2 = \text{Var}(S)$, he argues that for sufficiently large values of n , S is approximately normally distributed. Using the standard continuity correction, Thompson uses the approximation

$$\Pr(S = s) \cong \Pr(s - 0.5 < N(\mu, \sigma^2) < s + 0.5).$$

Calculating a midpoint approximation using a single subinterval, the *normal PDF approximation* is obtained, which is

$$\Pr(S = s) \cong \frac{1}{\sqrt{2\pi}\sigma} e^{-(s-\mu)^2/2\sigma^2}.$$

Instead of settling for approximations of the probabilities, we will show how APPL procedures, including **Convolution**, can retrieve exact solutions, while still giving the same insight into the problem.

Example 7.10. Suppose that X_1, X_2, \dots, X_{20} are independent and that $X_i \sim \text{Bernoulli}(p_i = \frac{29+2i}{100})$, $i = 1, 2, \dots, 20$. Let $S = \sum_{i=1}^{20} X_i$. (Here S denotes the total number of successes obtained in a series of independent trials where the probability of success varies from trial to trial.) Give an exact probability table for S for $s = 2, 3, \dots, 10$ [79, page 53].

Using Thompson's notation, $\mu = 10$ and $\sigma^2 = 2367/500$, and so $\Pr(S = s) \cong \frac{1}{\sqrt{2\pi}\sqrt{2367/500}} e^{-(s-10)^2/(2367/250)}$, $s = 0, 1, \dots, 20$.

Using APPL, we obtain the exact distribution of the random variable S by calling **Convolution** inside a Maple **for** loop:

```
> p := (29 + 2 * 1) / 100;
> S := BernoulliRV(p);
> for i from 2 to 20 do
>   p := (29 + 2 * i) / 100;
>   X := BernoulliRV(p);
>   S := Convolution(S, X);
> od;
> S;
```

Table 7.1 contains the exact probabilities found with APPL and the normal PDF approximations for $s = 2, 3, \dots, 10$.

Example 7.11. There are 20 girls and 30 boys in Group 1, 25 girls and 25 boys in Group 2, and 10 girls and 10 boys in Group 3. If 10 children are randomly chosen from each group and S denotes the total number of girls chosen, give an exact probability table for S for $s = 7, 8, \dots, 21$ [12, pages 53–54].

Let X_1, X_2 , and X_3 be the three independent hypergeometric random variables, and let $S = X_1 + X_2 + X_3$. The mean and variance of S are $\mu = E[S] = E[X_1] + E[X_2] + E[X_3] = 14$ and $\sigma^2 = \text{Var}(S) = \text{Var}(X_1) + \text{Var}(X_2) + \text{Var}(X_3) = 101/19$ (since X_1, X_2 , and X_3 are independent). Table 7.2 shows the normal PDF approximation values $\Pr(S = s) \cong \frac{1}{\sqrt{2\pi}\sqrt{101/19}} e^{-(s-14)^2/(202/19)}$ for $s = 7, 8, \dots, 20$.

Table 7.2. The exact probabilities and normal PDF approximations for $\Pr(S = s)$ for $s = 7, 8, \dots, 21$.

s	True $\Pr(S = s)$ from APPL	Approximation of true $\Pr(S = s)$	Normal PDF approximation
7	$\frac{4641594894759547665}{3082276280132202064912}$	0.0015	0.0017
8	$\frac{97479371530863990}{17512933409842057187}$	0.0056	0.0059
9	$\frac{12613791756912076515}{770569070033050516228}$	0.0164	0.0165
10	$\frac{74849525260411094591}{1926422675082626290570}$	0.0389	0.0384
11	$\frac{57967137494173367365}{770569070033050516228}$	0.0752	0.0742
12	$\frac{2096975232909133615}{17512933409842057187}$	0.1197	0.1188
13	$\frac{22076335771392253895}{140103467278736457496}$	0.1576	0.1575
14	$\frac{317244095646532855}{1843466674720216546}$	0.1721	0.1730
15	$\frac{9955623438355053449}{63683394217607480680}$	0.1563	0.1575
16	$\frac{217921905682010165}{1843466674720216546}$	0.1182	0.1188
17	$\frac{1894259194489549345}{25473357687042992272}$	0.0744	0.0742
18	$\frac{71588441634588035}{1843466674720216546}$	0.0388	0.0384
19	$\frac{10756216836381565}{641205799902684016}$	0.0168	0.0165
20	$\frac{1208983087163529637}{202781334219223820060}$	0.0060	0.0059
21	$\frac{280730797358534065}{162225067375379056048}$	0.0017	0.0017

The APPL code

```
> X := [[1 / 3, 1 / 3, 1 / 3], [-1, 0, 1],
        ["Discrete", "PDF"]];
> S := ConvolutionIID(X, 150);
> PDF(S, 5);
```

yields the exact PDF for S . The statement $\text{PDF}(S, 5)$ returns

$$\Pr(S = 5) = \frac{160709987007649212790999852367465829596098558279031212787052332840770}{4567759074507740406477787437675267212178680251724974985372646979033929},$$

which is approximately 0.03518. Since the support values of the parent population are -1 , 0 , and 1 , $\Pr(S = 5)$ can be computed using a combinatorics approach:

$$\Pr(S = 5) = \sum_{\substack{(p,q,r) | p+q+r=150, \\ 0 \leq p \leq 150, \\ 0 \leq q \leq 150, \\ 0 \leq r \leq 150, \\ r-p=5}} \binom{150}{p,q,r} \left(\frac{1}{3}\right)^p \left(\frac{1}{3}\right)^q \left(\frac{1}{3}\right)^r$$

or equivalently

$$\Pr(S = 5) = \sum_{p=0}^{72} \binom{150}{p, 145 - 2p, 5 + p} \left(\frac{1}{3}\right)^{150},$$

yielding the same result as the APPL code. The true power of APPL is not demonstrated in this particular example because of the special nature of the parent populations. The APPL approach allows for unequal mass values and more general support values. Also, more than three support values can be used in the APPL approach. Although a combinatorics method may still be used on these types of examples, the resulting expressions will not be as easy to derive or to evaluate as the expression obtained in the sum above.

The **Convolution** algorithm that was constructed (with heaps) to compute the PDF of the sum was extended to determine the PDF of the product also. Because of negative and positive support values for random variables X and Y , the product algorithm must split the array \hat{A} into four quadrants before forming heaps. The quadrants are split based on X 's and Y 's negative and non-negative support values using the support from Example 7.2. Figure 7.14 illustrates the evolution of the algorithm in the most general case when both X and Y have negative and positive support values. The product algorithm works from the corners of quadrants two and four toward the center where the quadrants meet, and then works outward from the center through quadrants one and three. For the product algorithm, the heap can receive values from two

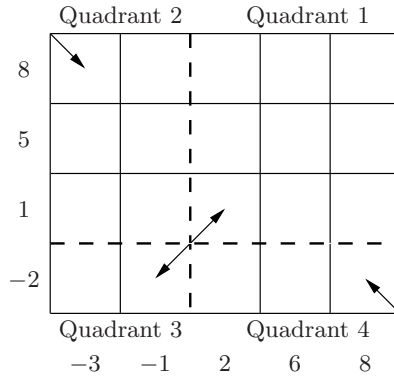


Fig. 7.14. Array A split into four quadrants for the **Product** procedure.

competing quadrants at a time. The **Product** procedure has been implemented for discrete random variables and is a part of APPL.

This concludes our discussion concerning the algorithm to calculate the distribution of the sum of two discrete random variables. The next chapter considers another non-trivial algorithm associated with discrete random variables, one which determines the distribution of order statistics drawn from discrete populations.

Order Statistics

This chapter presents an algorithm for computing the PDF of order statistics drawn from discrete parent populations, along with an implementation of the algorithm in APPL. Several examples illustrate the utility of this algorithm.

8.1 Notation and Taxonomy

Over a thousand references cited in the survey text on order statistics by David and Nagaraja [15] reflect the interest in the theory and application of order statistics. Many popular mathematical statistics texts, such as Hogg, et al. [37], Casella and Berger [12], and Rice [72], address only distributions of order statistics for iid random variables drawn from continuous populations due to the mathematical tractability of their theoretical formulas.

Results for order statistics drawn from discrete parent populations are sparse and usually specialized to fit one particular discrete population (e.g., Young [83], Srivastava [78], Ciardo et al. [13]). Arnold et al. [4] devote a chapter to discrete order statistics. We present an algorithm for determining distributions of order statistics drawn from a general discrete parent population. The algorithm presented in this chapter handles discrete parent populations with finite or infinite support, and sampling with or without replacement. Application areas, bootstrapping in particular, are postponed until Chapter 9.

Let X_1, X_2, \dots, X_n be iid discrete random variables defined on support Ω , each with CDF $F_X(x)$ and PDF $f_X(x)$. Let $X_{(1)} \leq X_{(2)} \leq \dots \leq X_{(n)}$ denote these random variables rearranged in weakly increasing order. Thus, $X_{(r)}$ is the r th smallest number in the sample, $r = 1, 2, \dots, n$, which is known as the r th “order statistic.” Because order statistics are random variables, it is possible to compute probabilities associated with values in their support. Since X_1, X_2, \dots, X_n is a random sample from a discrete population, the PDF of the r th order statistic cannot always be expressed as a single formula, as in the continuous case (see Section 3.2). When working with discrete random

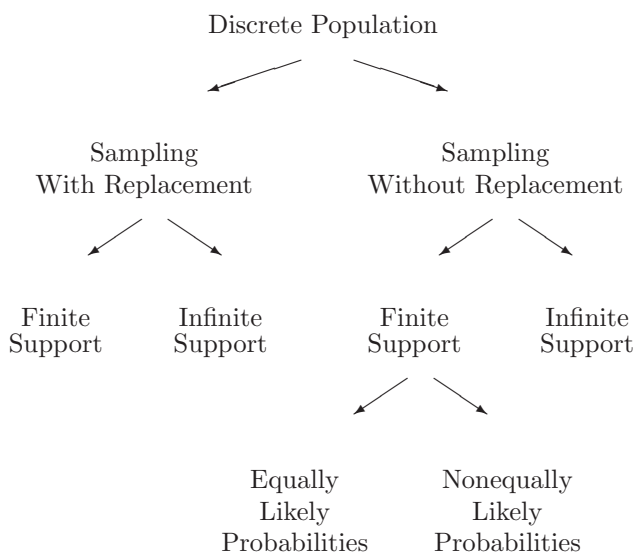


Fig. 8.1. Categorization of discrete order statistics by sampling convention, support, and probability distribution.

variables, the computation of the PDF of the r th order statistic will fall into one of several categories, depending on the sampling convention (with or without replacement), the random variable's support (finite or infinite), and the random variable's distribution (equally likely or nonequally likely probabilities). A taxonomy of these categories appears in Figure 8.1.

The APPL procedure `OrderStat(X, n, r, ["wo"])` has three required arguments: a random variable X , the number of items n randomly drawn from the population with PDF $f_X(x)$, and the index r of the desired order statistic. An optional fourth argument "wo" can be specified to indicate that the items are drawn from the population *without* replacement. `OrderStat` can be used for continuous and discrete populations; the algorithm associated with discrete populations is detailed in this chapter. The implementation steps of the `OrderStat` algorithm are explained in detail in the following two sections ("Without Replacement" and "With Replacement") and further details and applications are given in Evans et al. [24]. In both cases, the output of the algorithm is $f_{X_{(r)}}(x)$, the PDF of the r th order statistic, where n items have been sampled either with or without replacement from a population with PDF $f_X(x)$. Each of the five branches displayed in the tree in Figure 8.1 is discussed, followed by an example illustrating the particular sampling convention, support, and population PDF combinations. The particular branch from Figure 8.1 being illustrated in each example is given in *italics* at the beginning of the example.

8.2 Sampling Without Replacement

Providing the string "wo" as the fourth argument to the `OrderStat` procedure indicates that items are drawn from the discrete population without replacement. Random variables with finite or infinite support are handled by two separate techniques.

If the population distribution has finite support, the population size is denoted by N . To specify this support in a compact form, it is temporarily assumed to be (without loss of generality) the ordered set $\{1, 2, \dots, N\}$. For example, if

$$f_X(x) = \begin{cases} 0.5 & x = 7 \\ 0.2 & x = 11 \\ 0.3 & x = 15, \end{cases}$$

then $N = 3$, and the support of X is temporarily assumed to be $\{1, 2, 3\}$, instead of $\{7, 11, 15\}$. The original support is reinstated after the order-statistic probabilities are computed.

If the population with PDF $f_X(x)$ has equally likely probability values, e.g., rolling a fair die, then by combinatorial analysis the PDF of the r th order statistic when n items are sampled is (Arnold et al. [4, page 54])

$$f_{X_{(r)}}(x) = \frac{\binom{x-1}{r-1} \binom{N-x}{n-r}}{\binom{N}{n}} \quad x = r, r+1, \dots, r+N-n.$$

Example 8.1. (Sampling without replacement; finite support; equally likely probabilities) Draw 15 cards at random and without replacement from a deck of 25 cards numbered $1, 2, \dots, 25$. Find the probability that the card numbered 10 is the median of the cards selected (Hogg and Craig [36, page 231]).

Let $f_X(x) = \frac{1}{25}$ for $x = 1, 2, \dots, 25$. The population size is $N = 25$, the size of the sample is $n = 15$, and the sample median $X_{(8)}$ corresponds to $r = 8$. The probability that the sample median equals 10 is given by the above formula as

$$\begin{aligned} \Pr(X_{(8)} = 10) &= \frac{\binom{9}{7} \binom{15}{7}}{\binom{25}{15}} \\ &= \frac{1053}{14858} \\ &\cong 0.0709. \end{aligned}$$

In addition to just computing one probability value as obtained by hand above, the `OrderStat` algorithm returns the PDF of the 8th order statistic (when $n = 15$ items are sampled from the population without replacement), which is

$$f_{X_{(8)}}(x) = \frac{1}{3268760} \binom{x-1}{7} \binom{25-x}{7} \quad x = 8, 9, \dots, 18.$$

The APPL code to compute the PDF of the sample median and the probability that the median equals 10 is

```
> X := UniformDiscreteRV(1, 25);
> Y := OrderStat(X, 15, 8, "wo");
> PDF(Y, 10);
```

If X has finite support N and nonequally likely probabilities, there are three cases to consider when computing the PDF of the r th order statistic:

1. One item is sampled: $n = 1$. The PDF of the r th order statistic is the same as the population PDF; i.e., $f_{X_{(1)}}(x) = f_X(x)$ for $x = 1, 2, \dots, N$.
2. The entire population is sampled: $n = N$. The PDF of the r th order statistic is $f_{X_{(r)}}(x) = 1$ for $x = r$.
3. More than one item, but not the entire population, is sampled: $n = 2, 3, \dots, N - 1$. In this nontrivial case, an $n \times N$ array, *ProbStorage*, is defined that eventually contains the probabilities $f_{X_{(r)}}(x)$ for $x = 1, 2, \dots, N$ and $r = 1, 2, \dots, n$. The value of $f_{X_{(r)}}(x)$ is found in row r and column x of the array. For example, if $X \sim \text{binomial}(5, \frac{1}{4})$ and $n = 3$ items are sampled (without replacement) from this population, then the *ProbStorage* array is 3×6 and has the form displayed in Table 8.1. The algorithm's implementation requires use of two APPL combinatorial procedures, *NextCombination* and *NextPermutation*, which were specifically created for use in the *OrderStat* algorithm. Following is a brief description of the algorithm for the without-replacement/finite support/nonequally-likely-probabilities case for $n = 2, 3, \dots, N - 1$.

- (a) The first lexicographical combination of n items sampled from the sequence of integers 1 to N is formed; it is $\{1, 2, \dots, n\}$. For $X \sim \text{binomial}(5, \frac{1}{4})$, the first lexicographical combination of $n = 3$ items from $\{1, 2, \dots, 6\}$ is $\{1, 2, 3\}$.

Table 8.1. Initial form of *ProbStorage* array for $X \sim \text{binomial}(5, \frac{1}{4})$, where $n = 3$ items are sampled without replacement. Bullets denote positive probability values; zeros denote impossible situations.

$r \backslash x$	1	2	3	4	5	6
1	•	•	•	•	0	0
2	0	•	•	•	•	0
3	0	0	•	•	•	•

- (b) Given a combination consisting of n distinct integers, the algorithm generates all possible permutations of that combination and their corresponding probabilities. For the example $X \sim \text{binomial}(5, \frac{1}{4})$, the first combination is $\{1, 2, 3\}$ and its permutations are $[1, 2, 3]$, $[1, 3, 2]$, $[2, 1, 3]$, $[2, 3, 1]$, $[3, 1, 2]$, $[3, 2, 1]$. The probabilities obtained for each of these permutations, respectively, are $\frac{13286025}{150351872}$, $\frac{13286025}{204334592}$, $\frac{13286025}{119164928}$, $\frac{13286025}{110607872}$, $\frac{13286025}{197270528}$, and $\frac{13286025}{134730752}$. As an example, the probability of obtaining the permutation $[3, 1, 2]$ is computed as

$$f_X(3) \cdot \frac{f_X(1)}{1 - f_X(3)} \cdot \frac{f_X(2)}{1 - (f_X(3) + f_X(1))}$$

or

$$\frac{135}{512} \cdot \frac{\frac{243}{1024}}{1 - \frac{135}{512}} \cdot \frac{\frac{405}{1024}}{1 - (\frac{135}{512} + \frac{243}{1024})} = \frac{13286025}{197270528}.$$

- (c) After the probability of each permutation generated in step (b) is computed, each permutation is “rewritten” lexicographically to determine the corresponding order-statistic probabilities. The permutation $[3, 1, 2]$, for example, is rewritten as $[1, 2, 3]$. The algorithm then adds the probability of obtaining that particular permutation to the appropriate three cells in the *ProbStorage* array, which is denoted by $\sum(r, x) + q$ in Table 8.2. The (r, x) cell accumulates the probabilities of the various permutations for which the r th order statistic has the value x . For example, the final value in the $(1, 1)$ cell represents the probability that the first order statistic assumes the value one. In the $\text{binomial}(5, \frac{1}{4})$ example, the probability of obtaining the permutation $[3, 1, 2]$, $q = \frac{13286025}{197270528}$, is added to the *ProbStorage* cells $(1, 1)$, $(2, 2)$, and $(3, 3)$, as illustrated in Table 8.2.
- (d) After all $n!$ permutations of a given combination are exhausted, the procedure **NextCombination** determines the next lexicographical combination. Given this new combination, the algorithm repeats steps (b)

Table 8.2. Computation of *ProbStorage* array for $X \sim \text{binomial}(5, \frac{1}{4})$, where $n = 3$ items are sampled without replacement. Given the permutation $[3, 1, 2]$, its probability $q = \frac{13286025}{197270528}$ is added to the current probability sums $\sum(r, x)$ in cells $(1, 1)$, $(2, 2)$, and $(3, 3)$. Zeros denote impossible situations.

$r \backslash x$	1	2	3	4	5	6
1	$\sum(1, 1) + q$	$\sum(1, 2)$	$\sum(1, 3)$	$\sum(1, 4)$	0	0
2	0	$\sum(2, 2) + q$	$\sum(2, 3)$	$\sum(2, 4)$	$\sum(2, 5)$	0
3	0	0	$\sum(3, 3) + q$	$\sum(3, 4)$	$\sum(3, 5)$	$\sum(3, 6)$

and (c). This process iterates through the $\binom{N}{n}$ combinations of size n chosen from a population of size N .

Example 8.2. (Sampling without replacement; finite support; non-equally likely probabilities) Let X be a random variable with PDF

$$f_X(x) = \begin{cases} p_1 & x = 1 \\ p_2 & x = 10 \\ p_3 & x = 100 \\ p_4 & x = 1000, \end{cases}$$

where $p_1 + p_2 + p_3 + p_4 = 1$ and $p_i > 0, i = 1, 2, 3, 4$. Find the distribution of the median when $n = 3$ items are sampled without replacement from the population.

Since sampling is without replacement, the median can only assume the values $x = 10$ and $x = 100$. The APPL code to compute the distribution of the median is

```
> X := [[p1, p2, p3, p4], [1, 10, 100, 1000],
        ["Discrete", "PDF"]];
> OrderStat(X, 3, 2, "wo");
```

which returns the PDF

$$f_{X_{(2)}}(x) = \begin{cases} \frac{p_4 p_2 p_1}{(1-p_4)(1-p_2-p_4)} + \frac{p_4 p_2 p_1}{(1-p_4)(1-p_1-p_4)} + \\ \frac{p_4 p_2 p_1}{(1-p_2)(1-p_2-p_4)} + \frac{p_4 p_2 p_1}{(1-p_2)(1-p_1-p_2)} + \\ \frac{p_4 p_2 p_1}{(1-p_1)(1-p_1-p_4)} + \frac{p_4 p_2 p_1}{(1-p_1)(1-p_1-p_2)} + \\ \frac{p_3 p_2 p_1}{(1-p_3)(1-p_2-p_3)} + \frac{p_3 p_2 p_1}{(1-p_3)(1-p_1-p_3)} + \\ \frac{p_3 p_2 p_1}{(1-p_2)(1-p_2-p_3)} + \frac{p_3 p_2 p_1}{(1-p_2)(1-p_1-p_2)} + \\ \frac{p_3 p_2 p_1}{(1-p_1)(1-p_1-p_3)} + \frac{p_3 p_2 p_1}{(1-p_1)(1-p_1-p_2)} & x = 10 \\ \frac{p_4 p_3 p_2}{(1-p_4)(1-p_4-p_3)} + \frac{p_4 p_3 p_2}{(1-p_4)(1-p_2-p_4)} + \\ \frac{p_4 p_3 p_2}{(1-p_3)(1-p_4-p_3)} + \frac{p_4 p_3 p_2}{(1-p_3)(1-p_2-p_3)} + \\ \frac{p_4 p_3 p_2}{(1-p_2)(1-p_2-p_4)} + \frac{p_4 p_3 p_2}{(1-p_2)(1-p_2-p_3)} + \\ \frac{p_4 p_3 p_1}{(1-p_4)(1-p_4-p_3)} + \frac{p_4 p_3 p_1}{(1-p_4)(1-p_1-p_4)} + \\ \frac{p_4 p_3 p_1}{(1-p_3)(1-p_4-p_3)} + \frac{p_4 p_3 p_1}{(1-p_3)(1-p_1-p_3)} + \\ \frac{p_4 p_3 p_1}{(1-p_1)(1-p_1-p_4)} + \frac{p_4 p_3 p_1}{(1-p_1)(1-p_1-p_3)} & x = 100. \end{cases}$$

If the population distribution from which the sample is drawn has a countably infinite support, e.g., $\{1, 2, \dots\}$, then the pattern established for finding the PDF of the r th order statistic in the finite-support case will not work. At this time, beyond the trivial case when $n = 1$ item is sampled, where $f_{X_{(r)}}(x) = f_X(x)$ for $r = 1, 2, \dots, N$, the **OrderStat** algorithm computes the PDF of the minimum order statistic when at most $n = 2$ items are sampled without replacement from a discrete population with infinite support. The $n = 2$ case is considered in the following paragraph. Work on the **OrderStat** algorithm to incorporate the pattern for values of $n \geq 3$ is an open research question.

When $n = 2$ items are sampled without replacement, the probability that the minimum order statistic has value x , for $x = 1, 2, \dots$, is given by

$$\Pr(X_{(1)} = x) = \Pr(X_1 = x) \Pr(X_2 \geq x + 1 \mid X_1 = x) + \sum_{y=x+1}^{\infty} \Pr(X_1 = y) \Pr(X_2 = x \mid X_1 = y).$$

Thus, the PDF of $X_{(1)}$ when $n = 2$ items are sampled is

$$f_{X_{(1)}}(x) = f_X(x) \left[\frac{S_X(x+1)}{1 - f_X(x)} + \sum_{y=x+1}^{\infty} \frac{f_X(y)}{1 - f_X(y)} \right] \quad x = 1, 2, \dots,$$

where $S_X(x)$ is the survivor function defined by $S_X(x) = \Pr(X \geq x)$.

Example 8.3. (Sampling without replacement; infinite support) Let $X \sim \text{geometric}(\frac{1}{2})$. Find the probability that the minimum order statistic is five when $n = 2$ items are sampled without replacement from the population.

The PDF of the minimum order statistic as calculated using the **OrderStat** algorithm is

$$f_{X_{(1)}}(x) = \frac{\left(\frac{1}{4}\right)^x}{1 - \left(\frac{1}{2}\right)^x} + \frac{1}{2} \left(\frac{1}{2}\right)^{x-1} \sum_{y=x+1}^{\infty} \frac{1}{2} \frac{\left(\frac{1}{2}\right)^{y-1}}{1 - \frac{1}{2} \left(\frac{1}{2}\right)^{y-1}}$$

for $x = 1, 2, \dots$. By substitution, it can easily be determined that $\Pr(X_{(1)} = 5) \cong 0.0020$. The PDF of the first order statistic and the probability that it assumes the value of five can be computed in APPL with the following statements:

```
> X := GeometricRV(1 / 2);
> Y := OrderStat(X, 2, 1, "wo");
> evalf(PDF(Y, 5));
```


8.3 Sampling With Replacement

If the optional fourth argument "wo" is *not* provided to the `OrderStat` procedure, the items are assumed to be sampled from the discrete population with replacement. When sampling with replacement, the PDF $f_X(x)$, CDF $F_X(x)$, and SF $S_X(x)$ are needed to determine the distribution of the r th order statistic.

If the random variable X has finite support Ω , then without loss of generality we can assume that $\Omega = \{1, 2, \dots, N\}$. The PDF of $X_{(r)}$ when n items are sampled with replacement from this finite population is given by

$$f_{X_{(r)}}(x) = \begin{cases} \sum_{w=0}^{n-r} \binom{n}{w} [f_X(1)]^{n-w} [S_X(2)]^w & x = 1 \\ \sum_{u=0}^{r-1} \sum_{w=0}^{n-r} \binom{n}{u, n-u-w, w} [F_X(x-1)]^u [f_X(x)]^{n-u-w} [S_X(x+1)]^w & x = 2, 3, \dots, N-1 \\ \sum_{u=0}^{r-1} \binom{n}{u} [F_X(N-1)]^u [f_X(N)]^{n-u} & x = N. \end{cases}$$

This formula is not valid in the special case when the discrete population consists of only one item, that is, when $N = 1$. The PDF of the r th order statistic in this case is simply $f_{X_{(r)}}(1) = 1$.

The formula for the PDF of $X_{(r)}$ is a direct result of the following observation. For the r th order statistic to equal x , there must be u values less than or equal to $x-1$ and w values greater than or equal to $x+1$, where $u = 0, 1, \dots, r-1$ and $w = 0, 1, \dots, n-r$. The other $n-u-w$ values must be equal to x . The CDF procedure is used to determine the probability of obtaining a value less than or equal to $x-1$, the SF procedure is used to determine the probability of obtaining a value greater than or equal to $x+1$, and the PDF procedure is used to determine the probability of obtaining the value x . The multinomial coefficient expresses the number of combinations resulting from a specific choice of u and w .

Taking an example from Arnold et al. [4, page 43], let X be a discrete uniform random variable with PDF $f_X(x) = \frac{1}{4}$, $x = 1, 2, 3, 4$. Then the CDF and SF of X , respectively, are

$$F_X(x) = \begin{cases} 0 & x < 1 \\ \frac{\lfloor x \rfloor}{4} & 1 \leq x < 4 \\ 1 & x \geq 4 \end{cases} \quad S_X(x) = \begin{cases} 1 & x \leq 1 \\ 1 - \frac{\lfloor x \rfloor}{4} & 1 < x \leq 4 \\ 0 & x > 4. \end{cases}$$

Suppose $n = 5$ values are sampled with replacement from this population. To calculate $f_{X_{(2)}}(3)$, i.e., the probability that the second order statistic is $x = 3$, evaluate the sum

$$f_{X_{(2)}}(3) = \sum_{u=0}^1 \sum_{w=0}^3 \binom{5}{u, 5-u-w, w} [F_X(2)]^u [f_X(3)]^{5-u-w} [S_X(4)]^w.$$

The first term in the summation is the probability of drawing all threes. The second term, in which $u = 0$ and $w = 1$, is the probability of drawing four threes and a value greater than or equal to four (which can only be the value four in this example). This collection of four threes and one four can be drawn five different ways. The subsequent terms in the sum have similar meanings.

Example 8.4. (Sampling with replacement; finite support) A fair die is cast eight independent times. Find the PDF of the smallest of the eight numbers obtained, $X_{(1)}$ (Hogg and Craig [36, page 230]).

To compute the numeric PDF by hand,

$$f_{X_{(1)}}(x) = \sum_{w=0}^7 \binom{8}{w} \left(\frac{1}{6}\right)^{8-w} \left(1 - \frac{x}{6}\right)^w$$

is calculated for $x = 1, 2, \dots, 6$. For example, the probability that the first order statistic is $x = 4$ is

$$\begin{aligned} f_{X_{(1)}}(4) &= \sum_{w=0}^7 \binom{8}{w} \left(\frac{1}{6}\right)^{8-w} \left(\frac{1}{3}\right)^w \\ &= \frac{1}{1679616} + \frac{1}{104976} + \frac{7}{104976} + \frac{7}{26244} + \frac{35}{52488} + \\ &\quad \frac{7}{6561} + \frac{7}{6561} + \frac{4}{6561} \\ &= \frac{6305}{1679616} \\ &\cong 0.0038. \end{aligned}$$

Similar calculations for $x = 1, 2, \dots, 6$ yield the PDF of the first order statistic as

$$f_{X_{(1)}}(x) = \begin{cases} \frac{1288991}{1679616} & x = 1 \\ \frac{36121}{186624} & x = 2 \\ \frac{58975}{1679616} & x = 3 \\ \frac{6305}{1679616} & x = 4 \\ \frac{85}{559872} & x = 5 \\ \frac{1}{1679616} & x = 6. \end{cases}$$

The APPL code required to generate this PDF is

```
> X := UniformDiscreteRV(1, 6);
> OrderStat(X, 8, 1);
```

The `OrderStat` algorithm yields the same results, but in addition returns the PDF of the minimum order statistic as a polynomial:

$$f_{X_{(1)}}(x) = -\frac{1}{209952}x^7 + \frac{91}{419904}x^6 - \frac{889}{209952}x^5 + \frac{38675}{839808}x^4 \\ - \frac{63217}{209952}x^3 + \frac{496951}{419904}x^2 - \frac{543607}{209952}x + \frac{4085185}{1679616}$$

for $x = 1, 2, \dots, 6$.

If the support of X is countably infinite, i.e., $\Omega = \{1, 2, \dots\}$, then the calculation of the PDF of $X_{(r)}$ is similar to the finite-support case. The main difference is that the formula used in the finite-support case is now used for values of x that are unbounded:

$$f_{X_{(r)}}(x) = \begin{cases} \sum_{w=0}^{n-r} \binom{n}{w} [f_X(1)]^{n-w} [S_X(2)]^w & x = 1 \\ \sum_{u=0}^{r-1} \sum_{w=0}^{n-r} \binom{n}{u, n-u-w, w} [F_X(x-1)]^u [f_X(x)]^{n-u-w} [S_X(x+1)]^w & x = 2, 3, \dots \end{cases}$$

Because the formula assumes that the support of X is $\{1, 2, \dots\}$, the algorithm only works for distributions with infinite right-hand tails.

Since it is physically impossible to execute the formula for infinitely many values of x , a general expression for $f_{X_{(r)}}(x)$ in terms of x is obtained by taking advantage of Maple's ability to sum and simplify complicated symbolic expressions. Maple computes (in symbolic terms) the double summation of a multinomial coefficient multiplied by the product of the CDF, PDF, and SF raised to various powers.

Example 8.5. (Sampling with replacement; infinite support) Define a geometric random variable X with parameter p ($0 < p < 1$) to be the trial number of the first success in repeated independent Bernoulli(p) trials. The PDF of X is $f_X(x) = pq^{x-1}$, for $x = 1, 2, \dots$, where $q = 1 - p$. Margolin and Winokur [57] tabulated values for the mean and variance of the r th order statistic of a geometric distribution for samples of size $n = 1, 5(5) 20$ for $r = 1(1) 5(5) 20$, where $r = 1, 2, \dots, n$ and $q = 0.25(0.25) 0.75$. The values are calculated to two decimal places.

Margolin and Winokur's treatment of order statistics for the geometric distribution is based on the use of recurrence relations. The formulas they use to compute the first and second moments of the r th order

statistic of a geometric distribution (with parameter $p = 1 - q$) when n items are sampled (with replacement) are

$$E[X_{(r)}] = n \binom{n-1}{r-1} \sum_{j=0}^{r-1} \frac{(-1)^j \binom{r-1}{j}}{(n-r+j+1)(1-q^{n-r+j+1})}$$

and

$$E[X_{(r)}^2] = n \binom{n-1}{r-1} \sum_{j=0}^{r-1} \frac{(-1)^j \binom{r-1}{j} (1+q^{n-r+j+1})}{(n-r+j+1)(1-q^{n-r+j+1})^2}.$$

The **OrderStat** algorithm can easily produce the *exact value* of any of the rounded figures given in their tables. If $X \sim \text{geometric}(\frac{1}{4})$, for example, then the exact values of the mean and variance of the third order statistic when $n = 5$ items are sampled can be found with the following APPL code:

```
> X := GeometricRV(1 / 4);
> Y := OrderStat(X, 5, 3);
> Mean(Y);
> Variance(Y);
```

yielding

$$E[X_{(3)}] = \frac{3257984}{1011395} \cong 3.22$$

and

$$\text{Var}[X_{(3)}] = \frac{13665723739776}{5114599230125} \cong 2.67.$$

As shown in Example 8.5, the **OrderStat** procedure can be combined with the **Mean** and **Variance** procedures to compute the mean and variance for a much larger range of arguments. APPL might be used in this context to eliminate the need for many of the tables in the statistical literature. In addition, Maple can compute results using symbolic parameters. If $Y \sim \text{geometric}(p)$, for example, then the variance of the minimum order statistic when $n = 6$ items are sampled with replacement can be computed with the APPL statements

```
> X := GeometricRV(p);
> Y := OrderStat(X, 6, 1);
> Variance(Y);
```

which yield

$$\text{Var}[Y_{(1)}] = \frac{p^6 - 6p^5 + 15p^4 - 20p^3 + 15p^2 - 6p + 1}{p^7 - 6p^6 + 15p^5 - 20p^4 + 15p^3 - 6p^2}.$$

8.4 Extension

One natural extension to the **OrderStat** algorithm is a **RangeStat** algorithm, which finds the PDF of the range of a sample of n items drawn from a discrete population, either with or without replacement.

Let X be a discrete random variable with PDF $f_X(x)$ and CDF $F_X(x)$, and let Y be a random variable representing the range of X . We can assume (without loss of generality) that the support of X is $\{1, 2, \dots, N\}$, where N is a positive integer or infinity. If we are sampling with replacement (which implies that the n draws are independent), then the probability that Y assumes the value y , where $y = 0, 1, \dots, N - 1$, is given by Burr [10]:

$$\Pr(Y = y) = \begin{cases} \sum_{x=1}^N [f_X(x)]^n & y = 0 \\ \sum_{x=1}^{N-y} \{[\Pr(x \leq X \leq x+y)]^n - [\Pr(x+1 \leq X \leq x+y)]^n \\ - [\Pr(x \leq X \leq x+y-1)]^n + [\Pr(x+1 \leq X \leq x+y-1)]^n\} = \\ \sum_{x=1}^{N-y} \{[F_X(x+y) - F_X(x-1)]^n - [F_X(x+y) - F_X(x)]^n \\ - [F_X(x+y-1) - F_X(x-1)]^n + [F_X(x+y-1) - F_X(x)]^n\} & y = 1, 2, \dots, N-1. \end{cases}$$

Of the four terms being summed in the $y = 1, 2, \dots, N-1$ case, the first term is the probability that all sampled elements lie between x and $x+y$ inclusive. The second term removes the probability that these elements do not include x , because this would result in a range that is less than y . The third term removes the probability that these elements do not include $x+y$, because this would also result in a range that is less than y . The fourth term adds back the probabilities of those elements that include neither x nor $x+y$, which were removed by both the second and third terms.

In the without-replacement case, the **RangeStat** algorithm basically follows the same steps as the **OrderStat** algorithm, including use of the procedures **NextCombination** and **NextPermutation**. In the trivial case when the entire population is sampled, the PDF of the range Y is $f_Y(y) = 1$ for $y = N - 1$. If $n = 2, 3, \dots, N$, where N is a positive integer less than or equal to n , then a single-dimensional array of length $N - n + 1$ is defined that will eventually contain the values of $f_Y(y)$ for $y = n - 1, n, \dots, N - 1$. As in the **OrderStat** algorithm, the first lexicographical combination of n items sampled from the sequence of integers 1 to N is formed. Given such a combination, the algorithm generates all possible permutations of that combination. The probability for each permutation is calculated (as described in Section 8.1), and then the maximum and minimum values of that permutation

are determined. The permutation's range, which is the difference between its maximum and minimum values, is computed and then the appropriate array cell is incremented by that permutation's probability.

This concludes our survey of data structures and selected algorithms in APPL. The remaining portion of the monograph concerns the application of the language to problems in computational probability.

Reliability and Survival Analysis

The remaining chapters contain dozens of computational probability applications using APPL. The applications range in complexity from brief examples to results and algorithms requiring long derivations. This chapter surveys some applications in reliability and the closely-related field of survival analysis.

Section 9.1 presents four brief examples concerning the probabilistic analysis of systems of components: (1) analyzing a k -out-of- n system; (2) determining the mean time to failure of a parallel system with symbolic parameters; (3) determining the conditional survivor function for a system with a piecewise survival distribution; and (4) determining the probability of mission success for a unit and two cold standby units. Section 9.2 contains an algorithm for determining a lower confidence bound on the system reliability from binary failure data collected on each of the independent components in a coherent system via bootstrapping. Section 9.3 uses APPL to avoid the use of resampling in performing a bootstrap analysis in a survival analysis setting.

9.1 Systems Analysis

Reliability engineers often compare competing system designs by the use of probabilistic analysis. This section considers a series of four short examples that use APPL procedures to manipulate random variables associated with component lifetimes in order to compute measures of performance associated with the system, such as the mean time to failure or the system reliability. We begin with the analysis of a k -out-of- n system.

Example 9.1. Consider a 4-out-of-6 system that operates when four or more of its six components are functioning. Let the six component lifetimes be iid Weibull random variables with SF

$$S(t) = e^{-(5t)^2} \quad t \geq 0.$$

Use APPL to find

- (a) the hazard function of the time to system failure,
- (b) the mean time to system failure,
- (c) the variance of the time to system failure,
- (d) the probability that the system survives to time 0.1.

Let X denote the component lifetimes and T denote the system lifetime. The APPL code to calculate the four quantities of interest is

```
> X := WeibullRV(5, 2);
> T := OrderStat(X, 6, 3);
> HF(T);
> Mean(T);
> Variance(T);
> SF(T, 1 / 10);
```

The first statement defines the random variable X , which is associated with the lifetimes of the components, each with a Weibull time to failure with scale parameter 5 and shape parameter 2. Since this system has six components, the system failure occurs upon the third failure. Therefore, the system lifetime is the third order statistic of six independent Weibull random variables, denoted by T . The **HF** procedure returns the hazard function of the system time to failure, which is

$$h_T(t) = \frac{3000t(1 - e^{-25t^2})^2}{10e^{-50t^2} - 24e^{-25t^2} + 15} \quad t \geq 0.$$

The next statement computes the mean of the system time to failure, which is

$$E[T] = -\frac{12\sqrt{5\pi}}{25} + \frac{\sqrt{6\pi}}{6} + \frac{3\sqrt{\pi}}{4} \cong 0.1505.$$

The next statement computes the variance of the system time to failure, which is

$$\text{Var}[T] = \frac{37}{1500} - \frac{11287\pi}{6000} + \frac{4\pi\sqrt{30}}{25} + \frac{18\pi\sqrt{5}}{25} - \frac{\pi\sqrt{6}}{4} \cong 0.002003.$$

Finally, the probability that the system survives to time 0.1 is determined by the **SF** procedure, which yields

$$S_T(0.1) = 10e^{-3/2} - 24e^{-5/4} + 15e^{-1} \cong 0.8734.$$

Following each of the final three APPL statements with the Maple statement **evalf(%)**; computes the floating-point approximations associated with the exact values.

All of the parameters in Example 9.1 were numeric, which resulted, for example, in numeric values for the system mean time to failure and probability of system survival to time 0.1. A similar analysis can be applied to achieve a more general solution for a system with symbolic parameters, as illustrated in Example 9.2.

Example 9.2. Find the expected time to system failure of a parallel system, where each of the n components has an independent exponential(λ) lifetime.

As in the previous example, let X denote the component lifetimes and T denote the system lifetime. Since the reliability function of a parallel system of n identical components, each with reliability p , is

$$r(p) = 1 - (1 - p)^n$$

and the survivor function of an exponential random variable with failure rate λ is

$$S_X(t) = e^{-\lambda t} \quad t > 0,$$

the system survivor function is

$$S_T(t) = 1 - (1 - e^{-\lambda t})^n \quad t > 0.$$

The mean time to system failure is given by the expression

$$E[T] = \int_0^\infty (1 - (1 - e^{-\lambda t})^n) dt.$$

This quantity can be calculated, for instance when $n = 8$, with the APPL statements

```
> X := ExponentialRV(lambda);
> T := MaximumIID(X, 8);
> Mean(T);
```

which return a mean time to system failure of $E[T] = 761/(280\lambda)$. The `MaximumIID(X, 8)` procedure call is equivalent to `OrderStat(X, 8, 8)`. This solution did not require that the user know the functional form of the reliability function of a parallel system. If, on the other hand, the modeler did know the functional form of the reliability function, the solution can be checked using entirely Maple statements:

```
> assume(lambda > 0);
> s := 1 - (1 - exp(-lambda * t)) ^ 8;
> int(s, t = 0 .. infinity);
```

which also return a mean time to system failure of $E[T] = 761/(280\lambda)$. The advantage afforded by APPL is that the mathematical details associated with the derivation can be bypassed.

Examples 9.1 and 9.2 have considered components and systems with lifetimes characterized by having positive support values. Situations occasionally arise where the component lifetimes have varying support, perhaps due to a warranty or burn-in program. The next example considers a system which has varying supports, making the bookkeeping tedious. APPL relieves the modeler of the tedium of keeping track of the varying supports.

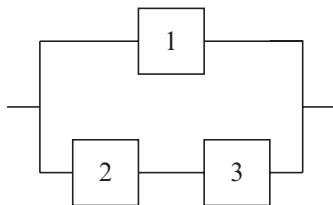


Fig. 9.1. Three-component system.

Example 9.3. Consider a three-component system with component 1 in parallel with the series arrangement of components 2 and 3, as shown in the block diagram in Figure 9.1. Let T_1 , T_2 , and T_3 be the independent lifetimes of the three components, where

$$T_1 \sim U(2, 10) \qquad T_2 \sim U(4, 14) \qquad T_3 \sim U(6, 18).$$

Find the 90th percentile of the distribution of the failure time of such a system that has survived to time 8.

Although this particular problem is conceptually straightforward, keeping track of the various supports for the components is tedious. Let $T = \max\{T_1, \min\{T_2, T_3\}\}$ denote the system failure time and let T_c denote system failure time conditioned on survival to time $t = 8$. The APPL code to solve this problem is given below.

```

> T1 := UniformRV(2, 10);
> T2 := UniformRV(4, 14);
> T3 := UniformRV(6, 18);
> T  := Maximum(T1, Minimum(T2, T3));
> TC := Truncate(T, 8, infinity);
> IDF(TC, 0.90);

```

This code returns the 90th percentile of the conditional distribution of the system lifetime as

$$\frac{32 - \sqrt{46}}{2} \cong 12.6088.$$

This code illustrates three new APPL procedures that have not yet been introduced. The first three statements define the distribution of T_1 , T_2 , and T_3 . The next statement calculates the PDF of T using the **Minimum** and **Maximum** procedures, which each take on two independent, but not necessarily identically distributed, random variables as arguments. The next statement uses the **Truncate** procedure to calculate the conditional distribution given survival to time 8. Finally, the **IDF** procedure is used to calculate the 90th percentile of T_c . The entire distribution of T_c has been computed in case one wants to perform further operations.

The APPL code given above is the ideal way to solve this particular problem. If one were uneasy about the correctness of the solution, it could be verified by Monte Carlo simulation. This Monte Carlo simulation is possible in APPL and Maple, although it is not computationally efficient in either language since Maple is an interpreted language. The Maple code below generates 10000 system lifetimes and estimates the quantity of interest. Surprisingly, Maple does not contain a random number generator for producing pseudorandom numbers. Instead, we use Maple's `rand` function, which generates a random 12-digit integer.

```
> _seed := 17:
> variates := []:
> for i from 1 to 10000 do
>   t1 := IDF(UniformRV(2, 10), rand() / 10 ^ 12):
>   t2 := IDF(UniformRV(4, 14), rand() / 10 ^ 12):
>   t3 := IDF(UniformRV(6, 18), rand() / 10 ^ 12):
>   t := max(t1, min(t2, t3)):
>   if (t > 8) then
>     variates := [op(variates), t]:
>   fi:
> od:
> variates := sort(variates):
> n := nops(variates):
> print(evalf(variates[floor(0.90 * n)]));
```

The first statement initializes the random number generator seed. The second statement initializes `variates` to an empty list. The `for` loop cycles through the 10000 replications of the simulation. The random variates $T_1 \sim U(2, 10)$, $T_2 \sim U(4, 14)$, and $T_3 \sim U(6, 18)$ are generated in the first three lines of code within the `for` loop using the `IDF` procedure. This is followed by the calculation of the (unconditional) system failure time $T = \max\{T_1, \min\{T_2, T_3\}\}$. If the value of T exceeds 8, then it is concatenated onto the right of the list `variates`. Outside of the `for` loop, the elements in `variates` are sorted using the Maple `sort` command. The `nops` function is used to find `n`, the number of elements in `variates`. Finally, the estimated 90th percentile of the conditional system lifetime is printed. When this code was executed five times (with `_seed` fixed prior to the first execution) the following five *estimates* for the conditional failure time were produced:

12.57 12.62 12.63 12.57 12.58.

The output of these five runs of Monte Carlo simulation hovers around the theoretical value of 12.61, which is the analytic solution provided by APPL. A more careful statistical analysis would involve calculating a confidence interval from the five estimates.

The three previous examples all considered systems with “warm” standby units, i.e., all components are on-line until failure. Some systems have cold standby units, as illustrated in the next example.

Example 9.4. Let T_1, T_2, T_3 be independent gamma random variables that represent the lifetime of a unit (T_1) and the lifetimes of two backup cold standby units (T_2 and T_3) taken on a mission of duration 10. If $T_1 \sim \text{gamma}(1/2, 3)$, $T_2 \sim \text{gamma}(1/4, 5)$, and $T_3 \sim \text{gamma}(1/6, 7)$, find the probability that the mission is successful.

The APPL code required to solve this problem is

```
> T1 := GammaRV(1 / 2, 3);
> T2 := GammaRV(1 / 4, 5);
> T3 := GammaRV(1 / 6, 7);
> T := Convolution(T1, T2, T3);
> SF(T, 10);
```

If there were more components with distributions following the same pattern and a Maple `for` loop was preferable, the following equivalent APPL code could be used, replacing the upper limit of the `for` by the number of components.

```
> T := GammaRV(1 / 2, 3);
> for i from 2 to 3 do
>   T := Convolution(T, GammaRV(1 / (2 * i), 2 * i + 1));
> od;
> SF(T, 10);
```

In either case, the probability of a successful mission is returned as

$$-\frac{57415240}{3}e^{-5/2} + \frac{2129286487}{256}e^{-5/3} + \frac{739}{256}e^{-5} \cong 0.9995.$$

If this probability of mission success is not high enough, more cold standby units could be added to the system.

9.2 Lower Confidence Bound on System Reliability

The examples to this point in the monograph have been drawn mainly from applied probability. This section takes up a problem of a statistical nature.

We consider the problem of determining a $(1 - \alpha)100\%$ lower confidence bound p_L on the system reliability for a coherent system of k components with independent lifetimes. Components of type i are subjected to a life test, $i = 1, 2, \dots, k$. The failure data is (y_i, n_i) , where y_i is the number of components of type i that pass the test and n_i is the number of components of type i being tested, $i = 1, 2, \dots, k$. We assume throughout that the components

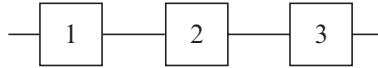


Fig. 9.2. Three-component series system.

Table 9.1. Failure data for a three-component series system example.

Component number	$i = 1$	$i = 2$	$i = 3$
Number passing (y_i)	21	27	82
Number on test (n_i)	23	28	84

fail independently, e.g., no common-cause failures. Bootstrapping is used to estimate a lower confidence bound on the system reliability.

A three-component ($k = 3$) series system is used as an example throughout this section, although the techniques described here apply to any coherent system of k independent components. A block diagram for the series system is given in Figure 9.2.

Binary data is collected on each component in the system. The number of components tested and the number of passes for each type of component for the example problem are given in Table 9.1.

The point estimate for the system reliability is

$$\frac{21}{23} \cdot \frac{27}{28} \cdot \frac{82}{84} = \frac{1107}{1288} \cong 0.8595.$$

There is no known exact confidence interval for this system reliability. We will consider only a *lower confidence bound* on the system reliability here, because this is typically the quantity of interest to a reliability engineer when assessing risk. The *bootstrapping* technique proposed by Efron and Tibshirani [19] can be used to calculate a lower 95% confidence interval bound. Other authors, such as Martin [60] and Padgett and Tomlinson [67], have proposed the use of bootstrapping for determining confidence limits. The approach used here differs conceptually from the standard application of bootstrapping, where the standard error of a *single* unknown distribution is estimated by resampling iid data. In our setting, there are k *different* distributions—one for each component—and we resample component reliabilities and combine using the reliability function to yield the system reliability.

Bootstrapping resamples some fixed number B of the systems, calculates the system reliability, then calculates the αB^{th} ordered system reliability. More specifically, for the three-component system of interest, the bootstrapping algorithm follows the following steps:

- For the first component, the data set (21 ones and 2 zeros) is sampled with replacement 23 times.
- These values are summed and divided by 23 yielding a reliability estimate for the first component.

Table 9.2. Bootstrap algorithm for calculating a $(1 - \alpha)100\%$ lower confidence bound for the reliability of a k -component series system.

for $j \leftarrow 1$ to B	resampling loop
for $i \leftarrow 1$ to k	loop through components
$\tilde{p}_i \leftarrow \text{binomial}(n_i, y_i/n_i)/n_i$	component i reliability
$z_j \leftarrow \prod_{i=1}^k \tilde{p}_i$	calculate system reliability
sort \mathbf{z}	sort the system reliability values
return $z_{\alpha B}$	return the estimate for the lower bound

- The previous two steps are repeated for components 2 and 3.
- The product of the reliability estimates for the three components are multiplied (because the components are arranged in series and their failures are independent) to give a system reliability estimate.

The four steps listed above are repeated B times. The B system reliability estimates are then sorted. Finally, the αB^{th} ordered system reliability is output, which is used as a lower bound on the system reliability.

The algorithm for estimating the $(1 - \alpha)100\%$ lower confidence interval bound is given in Table 9.2, where \tilde{p}_i is a bootstrap estimate for the reliability of component i and z_j is a bootstrap estimate of the system reliability. Indentation is used to indicate begin–end blocks in the pseudocode. Generating a binomial random variate is appropriate because the resampling from the data set is performed with replacement. The returned value $z_{\alpha B}$ is the order statistic associated with the z_j 's generated in the outside loop. See Law [48] for handling the case when αB is not an integer.

This algorithm has been implemented in S-Plus as a function named `seriessystemboot`, which is given below.

```

> seriessystemboot <- function(n, y, alpha) {
>   k <- length(n)
>   b <- 10000
>   z <- rep(1, b)
>
>   point <- prod(y) / prod(n)
>
>   for (j in 1:b) {
>     for (i in 1:k) {
>       z[j] <- z[j] * rbinom(1, n[i], y[i] / n[i]) / n[i]
>     }
>   }
>   z <- sort(z)
>   pl <- z[floor(alpha * b)]
>   c(point, pl)
> }

```

The first two arguments, `n` and `y`, are vectors of length k , and the third argument, `alpha`, is a real number between 0 and 1. Thus the S-Plus function call

```
> seriessystemboot(c(23, 28, 84), c(21, 27, 82), 0.05)
```

prints a point estimate and a 95% lower confidence interval bound on the system reliability for the three-component series system considered in this section. After a call to `set.seed(3)` to initialize the random number seed, five calls to `seriessystemboot` yielded the following estimates for p_L :

```
0.7426057    0.7482993    0.7456744    0.7486690    0.7453416.
```

The dispersion associated with these five estimates is due to the finite choice of B , i.e., $B = 10000$. Based on these results, the third digit after the decimal point can't be reported with certainty. A modeler would be uncertain how many digits to report on the estimate. Clearly a major drawback of this approach is our inability to determine the size of the resampling error as a function of the size of B .

Resampling error can be eliminated using APPL. The APPL statements given below utilize the `Product` and `Transform` procedures.

```
> X1 := BinomialRV(23, 21 / 23);
> X1 := Transform(X1, [[x -> x / 23], [0, infinity]]);
> X2 := BinomialRV(28, 27 / 28);
> X2 := Transform(X2, [[x -> x / 28], [0, infinity]]);
> X3 := BinomialRV(84, 82 / 84);
> X3 := Transform(X3, [[x -> x / 84], [0, infinity]]);
> T := Product(X1, X2, X3);
```

This alternative approach to determining a lower 95% bootstrap confidence interval bound for the system reliability is equivalent to using an infinite value for B . This is equivalent to deriving the entire PDF of the bootstrap estimate of the system reliability. Since \hat{p}_i can assume any one of $n_i + 1$ values, there are a possible $24 \cdot 29 \cdot 85 = 59160$ potential mass values for the random variable T determined by the two calls to the `Product` procedure. Of these, only 6633 are distinct because the `Product` procedure combines repeated values. Since the random variable T from the APPL code plays an analogous role to the vector \mathbf{z} from the bootstrap algorithm in Table 9.2, the lower 95% bootstrap confidence interval bound is the 0.05 fractile of the distribution of T , which is $p_L = 6723/9016 \cong 0.746$. This result using APPL is consistent with the standard resampling approach for finding the lower confidence interval limit based on the five results presented earlier (one equals 6723/9016 exactly, two are above 6723/9016, and two are below 6723/9016).

Bootstrapping worked well in this example, but will encounter problems when systems with perfect test results on one or more components are encountered. The problem that arises when component i has perfect test results (i.e.,

$y_i = n_i$) is that the sample size n_i becomes irrelevant. A test in which two components out of two pass the test is equivalent to one in which 100 components out of 100 pass the test from the perspective of the bootstrapping algorithm. Efforts to overcome the problem associated with perfect component test results by placing a beta conjugate prior distribution on component reliabilities are outlined in Leemis [51].

9.3 Survival Analysis

Survival analysis is a branch of statistics that concerns “time-to-event” data. For biological organisms, the interest might be in the time to death or the remission time of a cancer. In reliability, the interest might be in the time to failure or the time to repair for a system. In actuarial science, the interest might be in the time to death of the insured or the time to first claim on the insurance of a structure. In sociology, the interest might be in the duration of a strike or the time to commit another crime after release from prison in a recidivism study. In economics, survival analysis is known as duration analysis or duration modeling. The key element in a survival analysis study is a nonnegative random variable (typically the time to some event) with significant dispersion and oftentimes with an associated vector of covariates. Two popular textbooks in survival analysis are Lawless [49] and Kalbfleisch and Prentice [41].

In this section, we investigate bootstrapping in survival analysis. More specifically, we consider the simple question of comparing some aspect of two populations. Using Efron and Tibshirani’s [19] notation, this section again considers eliminating the generation of B bootstrap samples when performing a bootstrap analysis by calculating the exact distribution of the statistic of interest via APPL. There are several reasons for considering this approach:

- Although bootstrap methods are relatively easy to apply, determining the number of bootstrap repetitions, B , to employ is a common problem facing practitioners (Andrews and Buchinsky [3]). A practitioner needs to be concerned about problem-specific requirements for B , e.g., “ B in the range of 50 to 200 usually makes \hat{se}_{boot} a good standard error estimator, even for estimators like the median” (Efron and Tibshirani [19, page 14]) or “ B should be ≥ 500 or 1000 in order to make the variability of [the estimated 95th percentile] acceptably low” for estimating 95th percentiles (Efron and Tibshirani [19, page 275]). In fact, “one can obtain a ‘different answer’ from the same data merely by using different simulation draws if B is too small, but computational costs can be great if B is chosen to be extremely large” (Andrews and Buchinsky [2, page 23]). Because of this, Andrews and Buchinsky [2] introduced a three-step method to determine B to attain a specified level of accuracy.

- Exact values are always preferred to approximations. There is no point in adding resampling error to the already-existing sampling error unnecessarily.
- A bootstrapping novice can easily confuse the sample size n and number of bootstrap samples B . Eliminating the resampling of the data set B times simplifies the conceptualization of the bootstrap process.
- In many situations, computer time is saved using the exact approach.

By way of example, this section shows how the `OrderStat` and `RangeStat` procedures, along with other APPL procedures, can be used to perform exact bootstrap analysis. The use of these algorithms eliminates the resampling variability that is present in the usual bootstrap procedure. Exact results in the bootstrapping literature are limited: “The only exact solutions that are available for the bootstrap variance of an L -statistic (e.g., mean, trimmed mean, median, quick estimators of location and scale, upper and lower quartiles) are for the specific cases of the sample mean and sample median (when the sample size is odd)” (Hutson and Ernst [39, page 89]). The application areas that will be presented here are the estimation of standard errors for the median, mean, and range, and interval estimation for the range.

Estimation of Standard Errors. The standard error of the sample mean, s/\sqrt{n} , is useful when comparing means, but standard errors for comparing other quantities (e.g., fractiles) are often intractable. The following examples consider the estimation of standard errors associated with the rat-survival data given in Table 9.3 from Efron and Tibshirani [19, page 11]. Seven rats are given a treatment and their survival times, given in days, are shown in the first row of the table. Nine other rats constitute a control group, and their survival times are shown in the second row of the table.

In the first example, we compare the medians of the two populations using bootstrapping.

Example 9.5. Consider first the estimation of the standard error of the difference between the medians of the two samples. The standard bootstrap approach to estimating the standard error of the median for the treatment group is to generate B bootstrap samples, each of which consists of seven samples drawn with replacement from 16, 23, 38, 94, 99, 141, and 197. The sample standard deviation of the medians of these B bootstrap samples is an estimate of the standard error of the median. Using the S-Plus commands

Table 9.3. Rat-survival data.

Group	Data	n	Median	Mean	Range
Treatment	16, 23, 38, 94, 99, 141, 197	7	94	86.86	181
Control	10, 27, 30, 40, 46, 51, 52, 104, 146	9	46	56.22	136

Table 9.4. Bootstrap estimates of the standard error of the median.

	$B = 50$	$B = 100$	$B = 250$	$B = 500$	$B = 1000$	$B = +\infty$
Treatment	41.18	37.63	36.88	37.90	38.98	37.83
Control	20.30	12.68	9.538	13.10	13.82	13.08

```

> set.seed(1)
> tr <- c(16, 23, 38, 94, 99, 141, 197)
> medn <- function(x){quantile(x, 0.50)}
> bootstrap(tr, medn, B = 50)

```

yielded an estimated standard error of 41.18 for the treatment data with $B = 50$ bootstrap replicates. With the `set.seed` function used to call a stream number corresponding to the associated column, Table 9.4 shows the estimated standard errors for several B values. There is considerable resampling error introduced for smaller values of B . The $B = +\infty$ column of Table 9.4 corresponds to the *ideal bootstrap estimate of the standard error* of $\hat{\theta}$, or $\text{se}_{\hat{F}}(\hat{\theta}^*) = \lim_{B \rightarrow +\infty} \hat{\text{se}}_B$, to use the terminology and notation in Efron and Tibshirani [19, page 46].

We now consider the possibility of avoiding resampling altogether using APPL. An additional APPL procedure was constructed, namely `BootstrapRV(data)` (henceforth referred to as `BootstrapRV`), whose only argument is a list of n data values named `data`. The `BootstrapRV` procedure creates a discrete random variable X that can assume the provided data values, each with probability $\frac{1}{n}$. If we put the treatment data values 16, 23, 38, 94, 99, 141, and 197 in a list and provide this list to the `BootstrapRV` procedure, it creates a discrete random variable X that can assume these data values each with probability $\frac{1}{7}$. The following APPL code calculates the bootstrap standard deviation of the median:

```

> X := BootstrapRV([16, 23, 38, 94, 99, 141, 197]);
> Y := OrderStat(X, 7, 4);
> sqrt(Variance(Y));

```

The random variable X has a discrete uniform distribution with support values equal to the data values from the treatment data set. The random variable Y is the fourth order statistic (the median) in seven draws with replacement from the treatment group of the rat-survival data set. Since the median in a resampling experiment must take on one of the seven data values, the support of Y contains seven elements. APPL returns the distribution of the random variable Y as

$$f_Y(y) = \begin{cases} \frac{8359}{823543} & y = 16 \\ \frac{80809}{823543} & y = 23 \\ \frac{196519}{823543} & y = 38 \\ \frac{252169}{823543} & y = 94 \\ \frac{196519}{823543} & y = 99 \\ \frac{80809}{823543} & y = 141 \\ \frac{8359}{823543} & y = 197. \end{cases}$$

The last APPL statement takes the square root of the variance of Y , yielding the standard error as $\frac{2}{823543} \sqrt{242712738519382} \cong 37.8347$. This is the upper-right-hand entry in Table 9.4. In a similar fashion, the *ideal bootstrap estimate of the standard error* of the median can be calculated in the control case giving the lower-right-hand entry in Table 9.4 as $\frac{1}{387420489} \sqrt{25662937134123797402} \cong 13.0759$. Finally, note that the seemingly large difference between the two sample medians ($94 - 46 = 48$) is only $48/\sqrt{37.83^2 + 13.08^2} \cong 1.19$ standard-deviation units away from zero, indicating that the observed difference in the medians is not statistically significant. Had the standard bootstrap procedure been applied with $B = 50$ bootstrap replications, Table 9.4 indicates that the number of standard-deviation units would have been estimated to be $48/\sqrt{41.18^2 + 20.30^2} \cong 1.05$. Although the conclusion in this case is the same, the difference between using $B = 50$ and $B = +\infty$ could potentially result in different conclusions for the same data set.

Unlike the sample median, the standard error of the mean happens to be tractable. In spite of this fact, we continue with the previous analysis and attempt to compare the sample means to illustrate how other APPL procedures can be used for comparing means.

Example 9.6. In a similar fashion to the previous example involving medians, the following S-Plus code uses $B = 50$ bootstrap replications to estimate the standard error of the mean for the treatment group.

```
> set.seed(1)
> x <- c(16, 23, 38, 94, 99, 141, 197)
> bootstrap(x, mean, B = 50)
```

(The `mean` function is built into S-Plus.) This code returned 23.89, which is the upper-left-hand entry in Table 9.5. When this code is

Table 9.5. Bootstrap estimates of the standard error of the mean.

	$B = 50$	$B = 100$	$B = 250$	$B = 500$	$B = 1000$	$B = +\infty$
Treatment	23.89	24.29	23.16	24.36	23.75	23.36
Control	17.07	13.83	13.40	13.13	13.55	13.35

executed for other values of B and for the control group, we see the usual pattern of nonmonotonic convergence as B increases in Table 9.5, which is consistent with stochastic simulation.

We again consider the use of APPL to avoid resampling. The support of the distribution of the mean is much larger than the previous example because there are $7^7 = 823543$ different samples with replacement that could be drawn, resulting in hundreds (762, to be exact) of possible sample means. The APPL code to calculate the exact value of the standard error of the mean for the treatment group is

```
> X := BootstrapRV([16, 23, 38, 94, 99, 141, 197]);
> W := ConvolutionIID(X, 7);
> g := [[x -> x / 7], [-infinity, infinity]];
> Y := Transform(W, g);
> sqrt(Variance(Y));
```

After the bootstrap random variable X is created as in the previous example, the APPL procedure `ConvolutionIID` sums $n = 7$ of these iid X 's to create a random variable W . The APPL procedure `Transform` transforms the random variable W using the transformation $Y = W/n$ to calculate the PDF of the sample mean, which is

$$f_Y(y) = \begin{cases} 1/7^7 = 1/823543 & y = 16 \\ \binom{7}{1}/7^7 = 1/117649 & y = 17 \\ \binom{7}{2}/7^7 = 3/117649 & y = 18 \\ \binom{7}{3}/7^7 = 5/117649 & y = 19 \\ \binom{7}{1}/7^7 = 1/117649 & y = 134/7 \\ \binom{7}{4}/7^7 = 5/117649 & y = 20 \\ \vdots & \vdots \\ 1/7^7 = 1/823543 & y = 197, \end{cases}$$

Table 9.6. Bootstrap estimates of the standard error of the range.

	$B = 50$	$B = 100$	$B = 250$	$B = 500$	$B = 1000$	$B = +\infty$
Treatment	39.57	35.46	34.70	35.50	33.81	35.46
Control	36.70	29.84	30.95	30.07	30.14	31.48

and the standard error $\frac{2}{49}\sqrt{327649} \cong 23.3635$. This, of course, is equal to $\sqrt{\frac{n-1}{n}} \frac{s}{\sqrt{n}} = \sqrt{\frac{6}{7}} \frac{s}{\sqrt{7}}$, where s is the standard deviation of the treatment survival times. This fact is the fortunate consequence of the mathematical tractability of the standard error for means. Other, less fortunate situations can be handled in a similar manner.

Similar APPL code for the treatment case yielded an estimated standard error in the $B = +\infty$ case of $\frac{1}{27}\sqrt{129902} \cong 13.3489$. The difference between the treatment and control group means, which is $86.86 - 56.22 = 30.64$, is not statistically significant since it is only $30.64/\sqrt{23.36^2 + 13.35^2} \cong 1.14$ standard deviations away from zero.

The previous two examples have estimated the standard errors of measures of central tendency (the median and mean). Estimation of the standard error of a measure of dispersion, the sample range R , for the rat survival data will now be considered.

Example 9.7. Once again, the S-Plus code given below creates a bootstrap estimate of the standard error of the range in the treatment case with $B = 50$.

```
> set.seed(1)
> x <- c(16, 23, 38, 94, 99, 141, 197)
> bootstrap(x, range, B = 50)
```

Table 9.6 contains the standard error estimates for other values of B and for the control group.

To determine whether there is a statistically significant difference between the sample ranges for the treatment and control groups in the rat-survival data, we again use the APPL code below for estimating the standard error of the range for the treatment group.

```
> X := BootstrapRV([16, 23, 38, 94, 99, 141, 197]);
> Y := RangeStat(X, 7);
> sqrt(Variance(Y));
```

The bootstrap estimate of the standard error of the range R for the treatment case corresponding to $B = +\infty$ is $\frac{2}{16807}\sqrt{88781509983} \cong 35.4569$. The distribution of the sample range Y from bootstrapping returned by APPL is

$$f_Y(y) = \begin{cases} 1/117649 & y = 0 \\ 18/117649 & y = 5 \\ 18/117649 & y = 7 \\ 18/117649 & y = 15 \\ 276/117649 & y = 22 \\ 18/117649 & y = 42 \\ 276/117649 & y = 47 \\ 36/117649 & y = 56 \\ 276/117649 & y = 61 \\ 276/117649 & y = 71 \\ 1734/117649 & y = 76 \\ 1734/117649 & y = 78 \\ 6792/117649 & y = 83 \\ 276/117649 & y = 98 \\ 3468/117649 & y = 103 \\ 6792/117649 & y = 118 \\ 20010/117649 & y = 125 \\ 6792/117649 & y = 159 \\ 20010/117649 & y = 174 \\ 48828/117649 & y = 181. \end{cases}$$

Similar APPL statements for the control case yielded an estimated standard error for the range as $\frac{2}{4782969} \sqrt{5666287777334555} \cong 31.4762$. The observed difference in the ranges between the treatment and control groups ($181 - 136 = 45$) is not statistically significant since it is only $45/\sqrt{35.46^2 + 31.48^2} \cong 0.95$ standard deviations away from zero.

Confidence Intervals. A final example is presented here to show how one of the shortcomings of the standard bootstrap approach can be overcome by using the parametric bootstrap method. APPL is used in both settings to eliminate resampling error.

The previous three examples estimated the standard errors of measures of central tendency and a measure of dispersion. This example constructs two confidence intervals for the sample range of the rat-treatment group.

Example 9.8. Let R be the range of the $n = 7$ observations in the treatment group from Table 9.3. The APPL statements

```
> X := BootstrapRV([16, 23, 38, 94, 99, 141, 197]);
> Y := RangeStat(X, 7);
> IDF(Y, 0.025);
> IDF(Y, 0.975);
```

yield a 95% confidence interval of $76 < R < 181$. This confidence interval has the unappealing property that the point estimator, $\hat{R} = 181$, is also the upper limit of the confidence interval.

Trosset [80] suggested an alternative method for computing a confidence interval for the range R , which involves parametric bootstrapping. First, an exponential distribution with mean $1/\theta$ is fit to the control data using the APPL MLE (maximum likelihood estimator) procedure. The procedure identifies the parameter estimate for the distribution as $\hat{\theta} = 7/608$. The (continuous) distribution of the sample range of $n = 7$ observations drawn from an exponential population with parameter $\hat{\theta} = 7/608$ is then computed. The confidence interval is determined with the CDF procedure. The following APPL statements yield a 95% confidence interval for the range R for $n = 7$ samples as $68 < R < 475$.

```
> treatment := [16, 23, 38, 94, 99, 141, 197];
> X := ExponentialRV(theta);
> thetahat := op(MLE(X, treatment, [theta]));
> Y := ExponentialRV(thetahat);
> Z := RangeStat(Y, 7);
> IDF(Z, 0.025);
> IDF(Z, 0.975);
```

This section has considered the use of bootstrapping for simple comparisons between two populations. In conclusion, for moderate sample sizes and test statistics that are tractable for APPL, the exact approach to bootstrapping can eliminate resampling error and oftentimes reduce computation time.

This completes our initial chapter on APPL applications to reliability and survival analysis. The next chapter concerns applications of APPL in stochastic simulation.

Stochastic Simulation

This chapter considers applications from stochastic simulation. Section 10.1 considers tests of randomness. Section 10.2 investigates the use of computational probability in input modeling. Section 10.3 contains a development of an algorithm to find the distribution of the Kolmogorov–Smirnov goodness-of-fit test statistic in the all-parameters-known case.

10.1 Tests of Randomness

Since random numbers play a pivotal role in Monte Carlo simulation, discrete-event simulation, and bootstrapping, having random number generators with good statistical properties is essential. Some of the most common types of random number generators are in the class of Lehmer generators. These generators are based on Lehmer’s algorithm [54] which defines an iterative equation that can be used to produce a stream of random numbers. A Lehmer generator requires three input parameters: m , a , and z_0 . The *modulus* parameter m is a fixed large prime integer. The *multiplier* parameter a is a fixed positive integer less than m . The *initial seed* z_0 can be any positive integer less than m and can vary from experiment to experiment. Lehmer’s algorithm produces integers in the range $[1, m - 1]$ via the iterative equation $z_{i+1} = az_i \bmod m$ for $i = 0, 1, \dots$. For proper choices of a and m (see Banks et al. [6], Fishman [26], or Law [48]), the period length of the generator is $m - 1$. To convert the z_i ’s into $U(0, 1)$ pseudo-random variates, referred to here as simply “random numbers,” each is divided by the modulus m , $x_i = z_i/m$, yielding a rational number in the set $\{\frac{1}{m}, \frac{2}{m}, \dots, \frac{m-1}{m}\}$.

It is known that the numbers produced by a Lehmer generator are not truly random, but with carefully chosen m and a it is possible to generate output that will be “random enough” from a statistical point of view. Lehmer generators are also considered “good” generators because their output can be replicated, they’re portable, efficient, and thoroughly documented. However, a disturbing amount of regularity inherent in the output of these generators

was discovered by Marsaglia [58]. He observed a lattice structure when consecutive random numbers were plotted as overlapping ordered n -tuples [i.e., $(x_0, x_1, \dots, x_{n-1}), (x_1, x_2, \dots, x_n), \dots$]. We consider two types of testing for random numbers in this section. The first is the more traditional Kolmogorov–Smirnov goodness-of-fit test and the second is based on the distance between consecutive random number pairs.

Kolmogorov–Smirnov Goodness-of-Fit Test. Consider the use of the Kolmogorov–Smirnov test for assessing model adequacy (goodness of fit) for a Lehmer generator. Assume that $m = 2^{31} - 1 = 2147483647$, $a = 7^5 = 16807$, and $z_0 = 987654321$, which is a popular random number generator described in Park and Miller [68]. The first five random numbers generated are

$$\begin{array}{ccc} \frac{1605065384}{2147483647} & \frac{1791818921}{2147483647} & \frac{937423366}{2147483647} \\[1ex] \frac{1334477970}{2147483647} & & \frac{252032522}{2147483647} \end{array}$$

or, approximately

$$\begin{array}{ccc} 0.7474168 & 0.8343807 & 0.4365218 \\[1ex] 0.6214147 & & 0.1173618. \end{array}$$

Since these five data values are being evaluated for their uniformity, there should be a reasonable match between their empirical CDF and the CDF for a $U(0, 1)$ random variable. If we let the Maple list `Sample` contain the five random numbers generated above, then the APPL statements required to plot these two functions over the interval $(0, 1)$, shown in Figure 10.1, are

```
> n := 5;
> a := 7 ^ 5;
> seed := 987654321;
> m := 2 ^ 31 - 1;
> Sample := [];
> for j from 1 to n do
>   seed := a * seed mod m;
>   Sample := [op(Sample), seed / m]:
> od;
> U := UniformRV(0, 1);
> PlotEmpVsFittedCDF(U, Sample, [], 0, 1);
```

The five parameters that are required by the APPL plotting procedure `PlotEmpVsFittedCDF` are the random variable whose CDF is to be plotted, the data values [held in a list], the parameters associated with the random variable [empty in this case of $U(0, 1)$ random variable], and the optional lower and upper plotting limits.

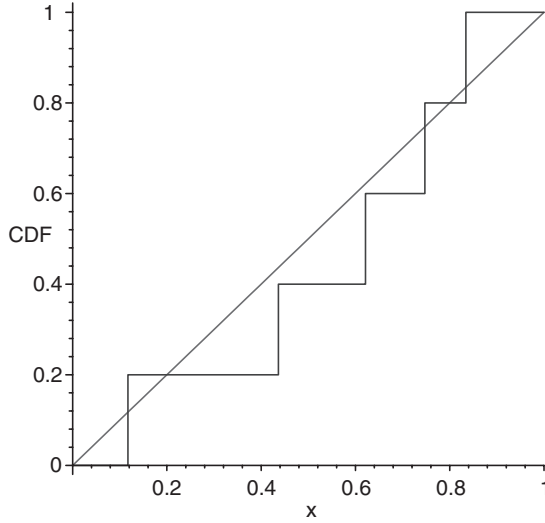


Fig. 10.1. The empirical CDF of `Sample` and the theoretical $U(0,1)$ CDF.

Let $F(x)$ be the hypothesized CDF and $F_5(x)$ be the empirical CDF. To determine the Kolmogorov–Smirnov test statistic,

$$D_5 = \sup_x |F(x) - F_5(x)|,$$

which measures the largest vertical distance between the two CDFs, requires the following additional command:

```
> TestStat := KSTest(U, Sample, []);
```

The approximate value of the test statistic for the five random numbers is 0.2365, which occurs just to the left of the random number 0.4365.

Since large values of the test statistic indicate a poor fit, and the CDF $F_{D_5}(x)$ of the test statistic is (see Section 10.3)

$$F_{D_5}(x) = \begin{cases} 0 & y < \frac{1}{10} \\ \frac{24}{625} (10x - 1)^5 & \frac{1}{10} \leq y < \frac{1}{5} \\ -288x^4 + 240x^3 - \frac{1464}{25}x^2 + \frac{672}{125}x - \frac{96}{625} & \frac{1}{5} \leq y < \frac{3}{10} \\ 160x^5 - 240x^4 + \frac{424}{5}x^3 + 12x^2 - \frac{168}{25}x + \frac{336}{625} & \frac{3}{10} \leq y < \frac{2}{5} \\ -20x^5 + 74x^4 - \frac{456}{5}x^3 + \frac{224}{5}x^2 - \frac{728}{125}x & \frac{2}{5} \leq y < \frac{1}{2} \\ 12x^5 - 6x^4 - \frac{56}{5}x^3 + \frac{24}{5}x^2 + \frac{522}{125}x - 1 & \frac{1}{2} \leq y < \frac{3}{5} \\ -20y^6 + 32y^5 - \frac{185}{9}y^3 + \frac{175}{36}y^2 + \frac{3371}{648}y - 1 & \frac{1}{2} \leq y < \frac{3}{5} \\ -8x^5 + 22x^4 - \frac{92}{5}x^3 + \frac{12}{25}x^2 + \frac{738}{125}x - 1 & \frac{3}{5} \leq y < \frac{4}{5} \\ 2x^5 - 10x^4 + 20x^3 - 20x^2 + 10x - 1 & \frac{4}{5} \leq y < 1 \\ 1 & y \geq 1, \end{cases}$$

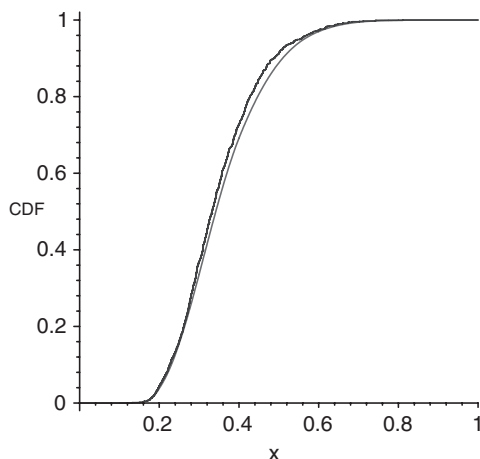


Fig. 10.2. Empirical CDF of 1000 Kolmogorov–Smirnov test statistics and the theoretical Kolmogorov–Smirnov CDF for $n = 5$.

the p -value for this particular test can be found with the additional APPL statement

```
> p := SF(KSRV(5), TestStat);
```

which yields $p \cong 0.8838$.

If this process is repeated for a total of 1000 groups of nonoverlapping consecutive sets of five random number pairs, the empirical CDF of the Kolmogorov–Smirnov statistics should be close to the theoretical from APPL if the random number generator is valid. Figure 10.2 is a plot of the empirical CDF of the 1000 Kolmogorov–Smirnov statistics versus the theoretical Kolmogorov–Smirnov CDF with $n = 5$. The empirical CDF lies slightly above the theoretical. If this experiment were performed repeatedly, the empirical CDFs should fluctuate around the theoretical CDF for a good random number generator.

Distance Between Consecutive Random Number Pairs. While it is true that the lattice structure discovered by Marsaglia [58] does not appear to be random at all, it may be the case that there is still a degree of randomness embedded within it. This randomness is not to be found in the structure of the lattice, however, but in the order in which the points are generated. More specifically, the empirical distribution of the distance between consecutive random number pairs should be close to the theoretical distance between consecutive random number pairs for a good random number generator. Duggan et al. [18] have developed a test of randomness based on the distance between consecutive random number pairs. We consider only the use of APPL to find the distribution of the distance between two of these pairs here. The derivations and analysis are an extension of the work by L’Ecuyer et al. [44] who test

for excessive clustering or repulsion of random points in space, where points X_i are non-overlapping k -dimensional vectors of successive output values of a random number generator. They consider the L_p -distances between all pairs of points generated X_i and X_j , whereas we only consider distances between consecutively generated points X_i and X_{i+1} , but treat both the overlapping and non-overlapping cases. Knuth [43] discusses specific tests that have traditionally been used to check the randomness of sequences, but all differ from our method. An ideal random number generator will have a close match between the empirical and theoretical distribution of the distance between consecutive random number pairs.

A random number generator should produce a stream of numbers that appears to be random despite the fact that they are generated by a deterministic function. Even though Marsaglia discovered that a plot of the n -tuples constructed from consecutive random numbers results in a lattice structure, the order in which the points are generated will affect the generator's statistical properties. That is, we hope to observe that points generally are not generated in order along a hyperplane nor is there necessarily a regular pattern of jumping between hyperplanes. With this in mind, we compare the distribution of the distances between consecutive points (ordered pairs) and the distribution of the distances between truly random points on the unit square. The assumption being made is that generators that yield a distribution of distances similar to the purely random points will be better random number generators.

When considering the distance between consecutive pairs of random numbers, the first question that must be addressed is whether to consider overlapping or non-overlapping ordered pairs. We choose the simpler case here, the non-overlapping ordered pairs (x_i, x_{i+1}) and (x_{i+2}, x_{i+3}) , because APPL can be used to find the theoretical distance between the points in 2-dimensional space. When working with non-overlapping pairs, it must be noted that analysis is only performed on half of the possible ordered pairs at a time and then merged together at the end. The two sets of ordered pairs that are examined are

$$\{(x_0, x_1), (x_2, x_3), (x_4, x_5), \dots, (x_{m-3}, x_{m-2})\}$$

and

$$\{(x_1, x_2), (x_3, x_4), (x_5, x_6), \dots, (x_{m-2}, x_{m-1})\},$$

where the first set consists of all ordered pairs with the first coordinate having an even index and the second set consists of all ordered pairs with the first coordinate having an odd index. The distances are calculated between consecutive points in the same set and then combined to form a single distribution.

Under the assumption that the pairs contain non-overlapping coordinates, the calculation of the theoretical distribution of the distances between consecutive pairs of randomly generated numbers is straightforward. If we

assume that X_1, X_2, X_3 , and X_4 are iid $U(0,1)$ random variables, then we must simply find the distribution of the distance between (X_1, X_2) and (X_3, X_4) given by

$$D = \sqrt{(X_1 - X_3)^2 + (X_2 - X_4)^2}.$$

Since D represents a distance between two points, it is obvious that $D > 0$ (the inequality is strict because the Lehmer algorithm cannot generate two identical pairs unless $m = 3$). Furthermore, since the longest distance possible is across the diagonal of the unit square, $D < \sqrt{2}$ (the inequality is also strict because the Lehmer algorithm cannot generate the numbers 0 or 1).

APPL provides an easy way to find the distribution of D which is a rather cumbersome hand calculation. The APPL code below calculates the PDF of D .

```
> U1 := UniformRV(0,1);
> U2 := UniformRV(0,1);
> V1 := Difference(U1, U2);
> g1 := [[x -> x ^ 2, x -> x ^ 2], [-infinity, 0, infinity]];
> V2 := Transform(V1, g1);
> V3 := Convolution(V2, V2);
> g2 := [[x -> sqrt(x)], [0, 2]];
> V4 := Transform(V3, g2);
```

The theoretical PDF of D is returned as

$$f(x) = \begin{cases} 2x(x^2 - 4x + \pi) & 0 < x \leq 1 \\ \frac{-2x \left(2\sqrt{x^2-1} + 4 - 4x^2 + 2\sqrt{x^2-1} \arcsin\left(\frac{x^2-2}{x^2}\right) + x^2\sqrt{x^2-1} \right)}{\sqrt{x^2-1}} & 1 < x < \sqrt{2}, \end{cases}$$

which is a continuous function since $\lim_{x \uparrow 1} f(x) = \lim_{x \downarrow 1} f(x) = 2(\pi - 3)$. The graph in Figure 10.3 is a plot of the PDF.

The corresponding CDF can be found with the additional CDF(V4); statement, yielding

$$F(x) = \begin{cases} \frac{1}{2}x^4 - \frac{8}{3}x^3 + \pi x^2 & 0 < x \leq 1 \\ \frac{\sqrt{x^2-1} \left(2 - 12x^2 \arcsin\left(\frac{x^2-2}{x^2}\right) - 3x^4 - 12x^2 \right) + 24x^2 - 24 + 16(x^2-1)^2}{6\sqrt{x^2-1}} & 1 < x < \sqrt{2}. \end{cases}$$

The distribution of D can then be used in the following fashion. Generate many non-overlapping pairs of random numbers, then calculate and store the distance between consecutive pairs. Perform a goodness-of-fit test to see if the distances provide a reasonable fit to the data values. If so, then the random number generator can be considered adequate. Details are given in Duggan et al. [18].

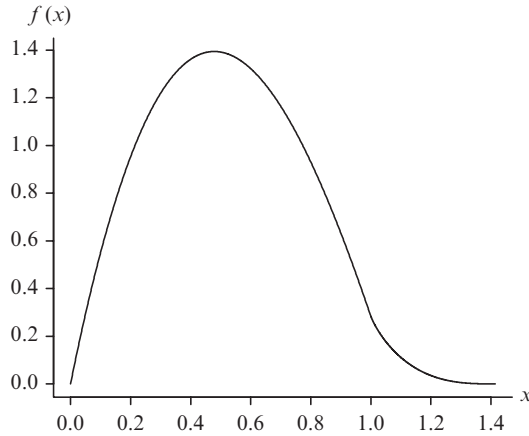


Fig. 10.3. The PDF of D .

10.2 Input Modeling

Stochastic simulation models have stochastic elements that mimic the probabilistic nature of the system under consideration. A close match between the input model and the true underlying probabilistic mechanism associated with the system is required for successful “input modeling.” The general question considered by input modeling is how best to model an element (e.g., arrival process, service times) in a discrete-event simulation given a data set collected on the element of interest. Oftentimes this involves fitting standard univariate parametric probability distributions to data sets. For brevity, it is assumed that data is available on the aspect of the simulation of interest. It is also assumed that raw data is available, as opposed to censored data, grouped data, or summary statistics. This example-driven section examines some applications of APPL for input modeling. Most simulation texts (e.g., Law [48]) have a broader treatment of input modeling than presented here. Nelson and Yamnitsky [63] survey advanced techniques.

Input modeling is typically performed using an input modeling package. These packages fit several distributions to a data set, then determine the distribution with the best fit by comparing goodness-of-fit statistics. But what if an appropriate input model is not included in one of these packages? The modeler must resort to deriving the appropriate estimators by hand for the appropriate input model. The purpose of this section is to investigate the use of APPL for input modeling. APPL allows an analyst to specify a standard or non-standard distribution for an input model, and have the derivations performed automatically. Input modeling serves as an excellent arena for illustrating the applicability and usefulness of APPL. Besides including pre-defined distributions for over 45 different types of continuous and discrete random variables and over 30 procedures for manipulating random variables

(e.g., **Convolution**, **Transform**), APPL contains input modeling procedures for parameter estimation, plotting empirical and fitted CDFs, and performing goodness-of-fit tests. Using examples, we illustrate its utility for input modeling.

Both APPL and Maple can be adapted for use in input modeling. This section gives several examples from Evans and Leemis [22] of cases where a symbolic language is of use in analyzing a data set.

Example 10.1. One of the tools for selecting a suitable input model is a plot of the coefficient of variation ($\gamma = \sigma/\mu$) versus the skewness

$$\gamma_3 = E \left[\left(\frac{X - \mu}{\sigma} \right)^3 \right]$$

for several popular input distributions. After constructing this plot, the sample coefficient of variation and the sample skewness can be plotted for a particular data set or data sets to determine an appropriate distribution for modeling the data. The Maple code that produces the plot in Figure 10.4 for the Weibull, gamma, log normal, and log logistic distributions uses the additional APPL procedures **CoefOfVar** and **Skewness**. The statements necessary to plot the gamma distribution's coefficient of variation versus skewness are shown below. The plots for the other distributions are calculated similarly. The Maple statement used to display all four plots in one graphic is also provided.

```
> unassign('kappa');
> lambda := 1;
```

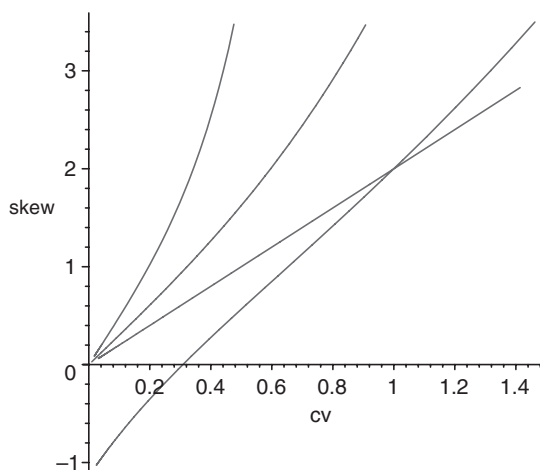


Fig. 10.4. Coefficient of variation γ vs. skewness γ_3 for the gamma, Weibull, log normal, and log logistic distributions.

```

> X := GammaRV(lambda, kappa);
> c := CoefOfVar(X);
> s := Skewness(X);
> GammaPlot := plot([c, s, kappa = 0 .. 750],
    labels = [gamma, gamma_3]):
.
.
.
> plots[display]({GammaPlot, WeibullPlot,
    LogNormalPlot, LogLogisticPlot},
    scaling = unconstrained);

```

The Maple `unassign` command is used to treat κ as a symbolic parameter.

Example 10.2. The following $n = 23$ ball bearing failure times (in millions of revolutions) are collected to determine an input model in a stochastic simulation:

105.84	28.92	98.64	55.56	128.04	45.60
67.80	105.12	48.48	51.84	173.40	51.96
54.12	68.64	93.12	68.88	84.12	68.64
41.52	127.92	42.12	17.88	33.00	

The original source for this data set is Lieblein and Zelen [56]. First, consider fitting an exponential distribution to this data set using maximum likelihood. The data set for the service times, `BallBearing`, is a pre-defined list in APPL. The APPL MLE procedure returns the maximum likelihood estimators as a list. Its arguments are the model, the data, and the parameters to be estimated. The APPL statements

```

> unassign('lambda');
> X := ExponentialRV(lambda);
> lamhat := MLE(X, BallBearing, [lambda]);

```

return $\hat{\lambda} \cong 0.0138$ as the maximum likelihood estimator. The additional APPL command

```

> PlotEmpVsFittedCDF(X, BallBearing,
    [lambda = lamhat[1]], 0, 180);

```

produces a plot of the empirical and fitted CDFs on one set of axes, as seen in Figure 10.5.

In order to assess the model adequacy, either a formal goodness-of-fit test can be performed, or goodness-of-fit statistics can be compared for competing models. The Kolmogorov–Smirnov test statistic, for example, can be computed with the additional APPL statement

```

> KSTest(X, BallBearing, [lambda = lamhat[1]]);

```

which returns 0.3068, indicating a rather poor fit.

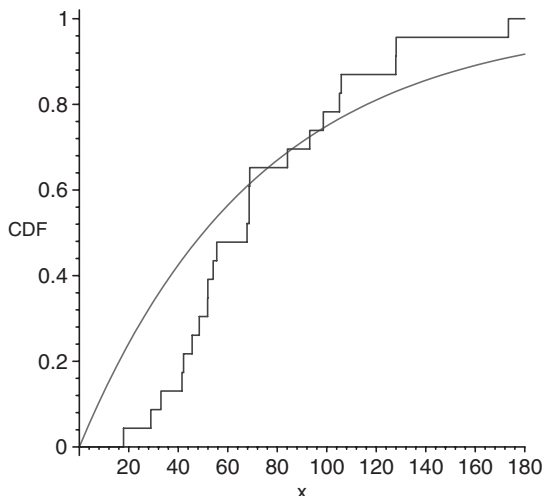


Fig. 10.5. Empirical and fitted exponential CDFs for the ball bearing data set.

As an alternative, one might consider fitting the *reciprocal* of an exponential random variable to the service times, as in the following example.

Example 10.3. Fit the reciprocal of an exponential random variable to the service times in the previous example.

The APPL statements required to find the distribution of the reciprocal of an exponential random variable and find the MLE for the unknown parameter are

```
> unassign('lambda');
> X := ExponentialRV(lambda);
> g := [[x -> 1 / x], [0, infinity]];
> Y := Transform(X, g);
> lamhat := MLE(Y, BallBearing, [lambda]);
```

which derive the PDF of Y to be

$$f_Y(y) = \frac{\lambda}{y^2} e^{-\lambda/y} \quad y > 0$$

and calculate the MLE $\hat{\lambda} \cong 55.06$. The function g is used to find the distribution of $Y = g(X) = 1/X$. As can be seen in Figure 10.6, the reciprocal of the exponential also provides a poor fit to the ball bearing data.

Neither the exponential model nor its reciprocal are appropriate for modeling the service times. It might be appropriate to consider two-parameter distributions as potential models, as shown in the next example.

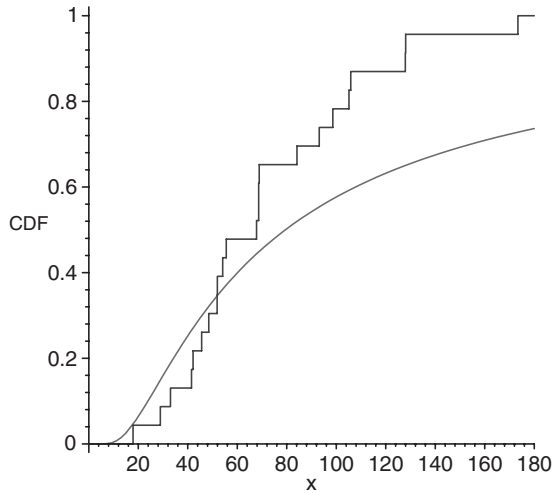


Fig. 10.6. Empirical and reciprocal exponential fitted CDFs for the ball bearing data set.

Example 10.4. Fit the inverse Gaussian and Weibull distributions to the data sets. Again using the APPL procedures MLE and KSTest,

```
> unassign('lambda');
> X := InverseGaussianRV(lambda, mu);
> hat := MLE(X, BallBearing, [lambda, mu]);
> KSValue := KSTest(X, BallBearing,
                    [lambda = hat[1], mu = hat[2]]);
```

yields an improved fit with $\hat{\lambda} \cong 231.67$, $\hat{\mu} \cong 72.22$, and a Kolmogorov–Smirnov test statistic of 0.088. The MLE procedure is able to return the appropriate values because the maximum likelihood estimators can be expressed in closed form for this particular distribution. Unfortunately, the statements

```
> unassign('lambda'); unassign('kappa');
> Y := WeibullRV(lambda, kappa);
> hat := MLE(Y, BallBearing, [lambda, kappa]);
```

fail to return the MLEs in APPL. The Maple numerical equation solving procedure `fsolve` is not clever enough to exploit some of the structure in the score vector that is necessary to find the MLEs. Therefore a special routine, `MLEWeibull`, has been written that computes MLEs for the Weibull distribution.

To summarize the ball bearing data set examples, Table 10.1 contains the Kolmogorov–Smirnov test statistic values for various distributions that were fit to the data via maximum likelihood estimation.

Table 10.1. Kolmogorov–Smirnov test statistics for various distributions.

Model	Test statistic
Exponential	0.307
Reciprocal of Exponential	0.306
Weibull	0.151
Gamma	0.123
Arctangent	0.093
Log normal	0.090
Inverse Gaussian	0.088

Another difficulty that can present itself in input modeling is the presence of censoring. A right-censored data set, for example, often occurs in reliability and biostatistical applications. Examples likely to arise in discrete-event input modeling situations include machine failure times (when some machines have not yet failed) and the analysis of rare events.

Example 10.5. Consider the problem of determining an input model for the remission time for the treatment group in the study concerning the drug 6-MP given by Gehan [27]. Letting an asterisk denote a right-censored observation, the remission times (in weeks) are

6 6 6 6* 7 9* 10 10* 11* 13 16
17* 19* 20* 22 23 25* 32* 32* 34* 35*.

Both `MP6` and `MP6Censor` are pre-defined lists in `APPL`. `MP6` is simply the 21 data values given above, and `MP6Censor` is the list

`[1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0]`

where 0 represents a right-censored observation and 1 represents an uncensored observation. The statements used to determine the MLE for an exponential distribution are

```
> unassign('lambda');
> X := ExponentialRV(lambda);
> hat := MLE(X, MP6, [lambda], MP6Censor);
```

The code yields the MLE for the failure rate as $\hat{\lambda} = 9/359$ remissions per week. Similarly, the statements

```
> unassign('lambda'); unassign('kappa');
> Y := WeibullRV(lambda, kappa);
> hat := MLEWeibull(MP6, MP6Censor);
```

yield the MLE estimates $\hat{\lambda} \cong 0.03$ and $\hat{\kappa} \cong 1.35$ for the Weibull distribution. The Kaplan–Meier product-limit survivor function estimate for the `MP6` data set, along with the fitted Weibull survivor

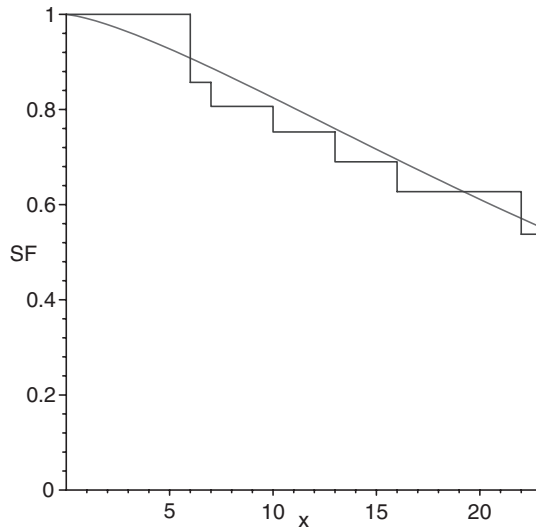


Fig. 10.7. Product-limit survivor function estimate and fitted Weibull survivor function for the 6-MP treatment group.

function are plotted in Figure 10.7 using the additional APPL statement

```
> PlotEmpVsFittedSF(Y, MP6,
                     [lambda = hat[1], kappa = hat[2]],
                     MP6Censor, 0, 23);
```

The downward steps in the estimated survivor function occur only at observed remission times. The six arguments required by the `PlotEmpVsFitted` procedure are the random variable whose SF is to be plotted, the data values in a list, the parameters associated with the random variable, the right-censoring vector in a list, and the lower and upper plotting limits. Note that the product-limit estimator cuts off after the largest observed remission time per the convention given in Lawless [49].

All of the input modeling examples thus far have been limited to continuous data. The next example fits the geometric distribution as a model for daily demand at a vending machine.

Example 10.6. A vending machine has capacity for 24 cans of “Purple Passion” grape drink. The machine is restocked to capacity every day at noon. Restocking time is negligible. The last five days have produced the following Purple Passion sales:

14 24 18 20 24.

The *demand* for Purple Passion at this particular vending machine can be estimated from the data by treating the 24-can sales figures as *right-censored* demand observations. If demand has the geometric distribution, with probability function

$$f(t) = p(1 - p)^t \quad t = 0, 1, 2, \dots,$$

find the MLE \hat{p} .

As discussed earlier, many procedures, like MLE, are able to handle discrete distributions. Since the pre-defined geometric distribution in APPL is parameterized for $t = 1, 2, \dots$, we need to define a geometric random variable with the different parameterization (used above) in the list-of-sublists data structure. The statements

```
> X := [[x -> p * (1 - p) ^ x],
        [0 .. infinity],
        ["Discrete", "PDF"]];
> PurplePass := [14, 24, 18, 20, 24];
> PurplePassCensor := [1, 0, 1, 1, 0];
> MLE(X, PurplePass, [p], PurplePassCensor);
```

yield $\hat{p} = 3/103$. Model adequacy is not considered for this particular example.

All previous examples have considered time-independent observations. There are occasions when a series of event times may be time dependent, and a more complicated input model may be appropriate.

Example 10.7. Ignoring preventive maintenance, ten odometer readings (from a certain model of car) associated with failures appearing over the first 100000 miles are, reading row-wise,

12942	17256	23254	28489	45561
85603	88143	94078	98231	99900.

Consider fitting a nonhomogeneous Poisson process to the above data set, where the ending time of the observation interval is assumed to be 100000 miles. The data can be approximated by a power law process (i.e., the intensity function has the same parametric form as the hazard function for a Weibull random variable). The following APPL statements, including the additional procedure MLENHPP, return $\hat{\lambda} \cong 0.000058$ and $\hat{\kappa} \cong 1.30$.

```
> unassign('lambda'); unassign('kappa');
> CarFailures := [12942, 17256, 23254, 28489, 45561,
                  85603, 88143, 94078, 98231, 99900];
> X := WeibullRV(lambda, kappa);
> hat := MLENHPP(X, CarFailures, [lambda, kappa]);
```

Every example considered thus far has used maximum likelihood to estimate the unknown parameters. APPL includes the procedure `MOM` for computing the method of moments estimators.

Example 10.8. Hurricanes typically strike the eastern and southern coastal regions of the United States, although they occasionally sweep inland before completely dissipating. The U.S. Weather Bureau reported that during the period from 1900 to 1969 a total of 36 hurricanes moved as far as the Appalachian Mountains. The maximum 24-hour precipitation levels (measured in inches) recorded from those 36 storms during the time they were over the mountains are (Larsen and Marx [47, pages 319–322])

31.00	2.82	3.98	4.02	9.50	4.50
11.40	10.71	6.31	4.95	5.64	5.51
13.40	9.72	6.47	10.16	4.21	11.60
4.75	6.85	6.25	3.42	11.80	0.80
3.69	3.10	22.22	7.43	5.00	4.58
4.46	8.00	3.73	3.50	6.20	0.67.

A histogram of the data, which can be plotted in Maple, suggests that the random variable X , which is the maximum 24-hour precipitation, might be well approximated by the gamma distribution. The following APPL code finds the method of moments estimates for the parameters λ and κ , where `Hurricane` is the above data set pre-defined in APPL:

```
> X := GammaRV(lambda, kappa);
> hat := MOM(X, Hurricane, [lambda, kappa]);
```

The resulting estimates for the parameters are $\hat{\lambda} = \frac{954000}{4252153} \cong 0.224$ and $\hat{\kappa} = \frac{6952275}{4252153} \cong 1.64$.

10.3 Kolmogorov–Smirnov Goodness-of-Fit Test

In this final section, we present an algorithm for computing the CDF of the Kolmogorov–Smirnov test statistic D_n in the all-parameters-known case. Birnbaum [9] gives an n -fold integral for the CDF of the test statistic which yields a function defined in a piecewise fashion, where each piece is a polynomial of degree n . Unfortunately, it is difficult to determine the appropriate limits of integration for computing these polynomials. Our algorithm performs the required integrations in a manner that avoids calculating the same integrals repeatedly, resulting in shorter computation time. It can be used to compute the entire CDF or just a portion of the CDF, which is more efficient for finding a critical value or a p -value associated with a hypothesis test. To date, critical tables have been approximated by various techniques including

asymptotic approximations, recursive formulas, and Monte Carlo simulation. Our approach yields exact critical values and significance levels. The algorithm has been implemented in APPL.

The Kolmogorov–Smirnov (KS) test statistic D_n is defined by

$$D_n = \sup_x |F(x) - F_n(x)|,$$

where n is the sample size, $F(x)$ is a hypothesized CDF with fixed parameters, and $F_n(x)$, also called the empirical distribution function, is a step-function that increases by $1/n$ at each data value. This statistic has been used for goodness-of-fit testing for continuous populations for decades. The KS test’s appeal is its straightforward computation of the test statistic and the distribution-free characteristic of D_n . Its drawback is that its CDF under the null hypothesis is difficult to determine, leaving one to calculate critical values with various approximation methods.

Birnbaum [9] gives the CDF of $D_n - \frac{1}{2n}$ as

$$\Pr\left(D_n < \frac{1}{2n} + v\right) = n! \int_{\frac{1}{2n}-v}^{\frac{1}{2n}+v} \int_{\frac{3}{2n}-v}^{\frac{3}{2n}+v} \cdots \int_{\frac{2n-1}{2n}-v}^{\frac{2n-1}{2n}+v} g(u_1, u_2, \dots, u_n) du_n \dots du_2 du_1$$

for $0 \leq v \leq \frac{2n-1}{2n}$, where

$$g(u_1, u_2, \dots, u_n) = 1$$

for $0 \leq u_1 \leq u_2 \leq \dots \leq u_n \leq 1$, and is 0 otherwise. Equivalently,

$$\Pr\left(D_n < \frac{1}{2n} + v\right) = n! \int_{\max(0, \frac{1}{2n}-v)}^{\min(1, \frac{1}{2n}+v)} \int_{\max(u_1, \frac{3}{2n}-v)}^{\min(1, \frac{3}{2n}+v)} \cdots \int_{\max(u_{n-1}, \frac{2n-1}{2n}-v)}^{\min(1, \frac{2n-1}{2n}+v)} 1 du_n \dots du_2 du_1.$$

Birnbaum’s rather harmless-looking integral is tedious to compute by hand, even for small values of n , due to the complexity of the region where g is nonzero. Evaluating his expression requires calculating many n -fold integrals whose limits are determined by considering a carefully chosen partition of the support of D_n . The difficult part of the process is setting the appropriate limits on these integrals. The final expression for the KS CDF is a piecewise polynomial function in which each polynomial has degree n .

The literature available on the KS statistic is extensive. Stephens’ article (Chapter 4 of D’Agostino and Stephens [14]) contains comprehensive coverage on the use of the KS statistic, as well as other statistics based on the empirical distribution function. He calculates the power of these goodness-of-fit tests. Johnson, Kotz, and Balakrishnan [40, page 460] consider this source to be so complete that they have deleted KS discussions in their second edition and

refer the reader to that compendium instead. For computing critical points of the KS distribution, we find five-digit accuracy as early as 1956 in Miller [61]. Miller relies on asymptotic results that converge fairly quickly to produce these estimates. Birnbaum's article [9] also explains how various sets of recursive formulas can be used to calculate certain critical points to reasonable levels of accuracy. Law [48, pages 346–351] has a well-written introduction to the KS test and indicates that critical points can only be calculated via computer techniques. There appears to be no source that produces exact distribution functions for any distribution where $n > 3$ in the literature. Birnbaum [9, page 144] gives the CDF of $D_n - \frac{1}{2n}$ for $n = 2$ and $n = 3$. Knuth [43] provides a functional form for the CDFs for the two statistics D_n^+ and D_n^- , but does not provide the CDF for $D_n = \max\{D_n^+, D_n^-\}$, a harder problem given the dependence between the two random variables. As a consequence of the apparent complexity of the required integration and lack of literature on exact distributions, we believed early on that a computational algebra system, such as Maple, would be necessary to compute the polynomials needed for the distribution of the KS test statistic.

Computing the Distribution of D_n . When $0 < v < \frac{1}{2n}$, Birnbaum's integral is easy to compute since none of the intervals of integration overlap. Additionally, these intervals are all wholly contained within the interval from 0 to 1. For example, u_1 varies over the interval from $\frac{1}{2n} - v$ to $\frac{1}{2n} + v$, and u_2 varies over the interval from $\frac{3}{2n} - v$ to $\frac{3}{2n} + v$. The interval for u_1 begins to the right of 0 and ends before the interval for u_2 begins. Thus when $0 < v < \frac{1}{2n}$, the limits of integration of Birnbaum's integral guarantee that $0 \leq u_1 \leq u_2 \leq \dots \leq u_n \leq 1$. Because the u_i 's are constrained in this way, we may replace Birnbaum's integrand $g(u_1, u_2, \dots, u_n)$ with 1, and our computation of the KS CDF requires only a single n -fold integral:

$$\begin{aligned} \Pr\left(D_n < \frac{1}{2n} + v\right) &= n! \int_{\frac{1}{2n}-v}^{\frac{1}{2n}+v} \int_{\frac{3}{2n}-v}^{\frac{3}{2n}+v} \dots \int_{\frac{2n-1}{2n}-v}^{\frac{2n-1}{2n}+v} 1 \, du_n \dots du_2 du_1 \\ &= n!(2v)^n \qquad 0 < v < \frac{1}{2n}. \end{aligned}$$

When v has a fixed value greater than $\frac{1}{2n}$, it is also desirable to replace Birnbaum's integrand by 1. In order to permit this replacement, we must only allow integration over that region of n -dimensional space for which $0 \leq u_1 \leq u_2 \leq \dots \leq u_n \leq 1$. Since the intervals of integration specified in Birnbaum's integral for different u_i 's can overlap, the smallest allowable value for any u_i is influenced by all u_j 's with subscripts less than i that can take on values in u_i 's interval of integration. This overlapping requires partitioning the interval from $\frac{1}{2n} - v$ to $\frac{2n-1}{2n} + v$ into subintervals (which we will henceforth refer to as u -subintervals), with the first u -subinterval starting at $\frac{1}{2n} - v$ and a new u -subinterval beginning whenever the left endpoint of one of Birnbaum's intervals of integration is encountered. When any u_i lies in a u -subinterval that consists entirely of negative values, Birnbaum's integrand is zero. For

this reason, only u -subintervals having a positive right endpoint contribute to the KS CDF.

Of course, the number of u -subintervals having a positive right endpoint depends on the value of v . For small values of v (i.e., $0 < v < \frac{1}{2n}$), $n + 1$ u -subintervals have a positive right endpoint since all n of Birnbaum's lower limits of integration are positive. For large values of v (i.e., $\frac{2n-3}{2n} < v < \frac{2n-1}{2n}$), only two u -subintervals have a positive right endpoint since only u_n 's lower limit of integration is positive. Intermediate values of v produce all possibilities from 2 to $n + 1$ u -subintervals having a positive right endpoint.

Because different values of v can yield different numbers of u -subintervals having a positive right endpoint, the interval $0 < v < \frac{2n-1}{2n}$ must be subdivided at the following values of v : $\frac{1}{2n}, \frac{3}{2n}, \frac{5}{2n}, \dots, \frac{2n-3}{2n}$. When the values of v remain within one of the resulting subintervals for v , the number of u -subintervals having a positive right endpoint will remain fixed.

Another complication arises because it is necessary to know, for a fixed value of v and for the u -subintervals produced by this value, which variables of integration u_i can take on values in each of the u -subintervals of $[0, 1]$. The previous subdivision of the values of v is not fine enough to allow unambiguous specification of the range for each u_i . When $i < j$, u_i and u_j have overlapping intervals of integration if the upper integration limit for u_i exceeds the lower integration limit for u_j , i.e., $\frac{2i-1}{2n} + v \geq \frac{2j-1}{2n} - v$. As a result, as v increases from 0 to $\frac{2n-1}{2n}$, new overlaps take place when v equals $\frac{1}{2n}, \frac{2}{2n}, \frac{3}{2n}, \dots, \frac{n-1}{2n}$. The interval $0 < v < \frac{2n-1}{2n}$ must be divided into subintervals at these values of v as well as at the values of v listed previously. We will henceforth refer to the resulting subintervals as v -subintervals.

Indicator matrices will be used to summarize the interrelationships between the possible values for the variables of integration u_1, u_2, \dots, u_n . For a fixed n and for values of v in the k th v -subinterval of $(0, \frac{2n-1}{2n})$, the indicator matrix A_k will show, by the presence of a 1 or a 0 in row i and column j , whether or not u_i can take on values in the j th u -subinterval. Finally, by defining paths through these indicator matrices, we will produce a complete set of n -fold integrals which satisfy the requirement $0 \leq u_1 \leq u_2 \leq \dots \leq u_n \leq 1$, and which produce the KS CDF when summed.

The algorithm for computing the CDF of D_n is divided into four phases. In Phase 1, an appropriate partition of the support of $D_n - \frac{1}{2n}$ is determined. In Phase 2, we define matrices A_k that are instrumental in determining the limits of integration in the n -fold integrals. In Phase 3, these integrals and their associated limits are computed. To take advantage of the similarities in the limits of integration of these n -fold integrals, they are grouped for efficiency and all evaluated to the same level of integration before proceeding to the next level. Finally, Phase 4 consists of translating the support of $D_n - \frac{1}{2n}$ to the support of D_n . Thus the input to the algorithm is a positive integer n and the output is the piecewise CDF of D_n .

Phase 1: Partition the Support of $D_n - \frac{1}{2n}$. The endpoints of the segments that define the support of the KS test statistic can be determined from the limits of integration given by Birnbaum. Using Birnbaum's formula, the *baseline* lower and upper limits of integration associated with $v = 0$ are

$$\left\{ \frac{1}{2n}, \frac{3}{2n}, \frac{5}{2n}, \dots, \frac{2n-1}{2n} \right\}.$$

As v increases, the support of $D_n - \frac{1}{2n}$ is broken into disjoint v -subintervals. The endpoints of these support v -subintervals consist of $v = 0$ and the values of v for which the endpoints of the n intervals of integration either:

- equal 0 or 1, *or*
- coincide.

The 0 or 1 cases occur at v values in the set

$$\left\{ \frac{1}{2n}, \frac{3}{2n}, \frac{5}{2n}, \dots, \frac{2n-1}{2n} \right\}.$$

The set of values of v for which the endpoints of different intervals of integration coincide are

$$\left\{ \frac{1}{2n}, \frac{2}{2n}, \frac{3}{2n}, \dots, \frac{n-1}{2n} \right\}.$$

Thus the union of these two sets and 0 comprise the endpoints of the v -subintervals of the support of $D_n - \frac{1}{2n}$.

The first phase of the algorithm given at the end of this section computes the above endpoints $v_0 = 0, v_1, v_2, \dots, v_m$, where m is the number of v -subintervals on which the CDF of $D_n - \frac{1}{2n}$ is defined. For any n ,

$$m = \left\lceil \frac{3n}{2} \right\rceil - 1.$$

Phase 2: Define the A Matrices. At this point in the algorithm, v_0, v_1, \dots, v_m are the endpoints of the m v -subintervals on which the CDF of $D_n - \frac{1}{2n}$ is defined.

Two book-keeping steps are needed in this phase of the algorithm. They are:

1. Define c_1, c_2, \dots, c_m as the midpoints of the v -subintervals of support for $D_n - \frac{1}{2n}$.
2. Define x_1, x_2, \dots, x_n as the midpoints of the intervals of integration for u_1, u_2, \dots, u_n in Birnbaum's n -fold integral. Thus

$$x_i = \frac{2i-1}{2n} \quad i = 1, 2, \dots, n.$$

Part of the algorithm involves defining the $n \times n$ indicator matrices A_1, A_2, \dots, A_m corresponding to the v -subintervals which form the support of $D_n - \frac{1}{2n}$. The rows of an A matrix refer to the variables of integration u_1, u_2, \dots, u_n . The columns of an A matrix refer to the u -subintervals, with the j th column corresponding to the interval from $x_j - v$ to $x_{j+1} - v$, except for the n th column, which corresponds to the interval from $x_n - v$ to $x_n + v$. If the (i, j) element of an A matrix equals 1, then the range of u_i includes at least part of the j th u -subinterval.

Since the (n, n) element of each A matrix is the only element in the n th row that is 1, the integrals that need to be computed for each segment of support of $D_n - \frac{1}{2n}$ can be visualized as a path in A , consisting of a sequence of moves from the (n, n) position of A to a nonzero element in the first row of A . All moves in the A matrix from row i to row $i - 1$ (for $i = n, n - 1, \dots, 2$) require that the following conditions are met:

1. The move must be from one of the 1's in row i to one of the 1's in row $i - 1$.
2. The move is either directly vertical or vertical and one or more spaces to the left.

That is, if the move begins at a 1 element in column j of row i , it must end at a 1 element in column 1 through j of row $i - 1$.

Example 10.9. For $n = 3$ and $k = 3$, the CDF is defined on the v -subinterval $\frac{2}{6} < v < \frac{3}{6}$ and has the following A_3 matrix:

$$A_3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

There are five different paths from the $(3, 3)$ element to the top row of the matrix. In the five matrices below, these five paths to the top row of the A matrix are displayed by using boldface type on the path. Next to each matrix is the corresponding triple integral that the path represents.

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & 1 \\ 0 & 0 & \mathbf{1} \end{bmatrix} \quad \int_0^{\frac{3}{6}-v} \int_{\frac{3}{6}-v}^{\frac{5}{6}-v} \int_{\frac{5}{6}-v}^1 1 \, du_3 \, du_2 \, du_1$$

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & 1 \\ 0 & 0 & \mathbf{1} \end{bmatrix} \quad \int_{\frac{3}{6}-v}^{\frac{5}{6}-v} \int_{u_1}^{\frac{5}{6}-v} \int_{\frac{5}{6}-v}^1 1 \, du_3 \, du_2 \, du_1$$

$$\begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ 0 & 1 & \mathbf{1} \\ 0 & 0 & \mathbf{1} \end{bmatrix} \quad \int_0^{\frac{3}{6}-v} \int_{\frac{5}{6}-v}^{\frac{3}{6}+v} \int_{u_2}^1 1 \, du_3 \, du_2 \, du_1$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\int_{\frac{3}{6}-v}^{\frac{5}{6}-v} \int_{\frac{5}{6}-v}^{\frac{3}{6}+v} \int_{u_2}^1 1 \, du_3 \, du_2 \, du_1$$

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\int_{\frac{5}{6}-v}^{\frac{1}{6}+v} \int_{u_1}^{\frac{3}{6}+v} \int_{u_2}^1 1 \, du_3 \, du_2 \, du_1$$

When these five integrals are summed, Birnbaum’s expression reduces to

$$\Pr\left(D_3 < \frac{1}{6} + v\right) = -4v^3 + \frac{11}{3}v - \frac{11}{27}$$
$$\frac{2}{6} < v < \frac{3}{6}.$$

Note that the innermost integral

$$\int_{u_2}^1 1 \, du_3$$

on the third, fourth, and fifth triple integrals are identical. The number of identical integrals of this type grows rapidly as n increases. The algorithm we have developed avoids recalculation of duplicate integrals. Continuing in this fashion for the other A matrices (i.e., A_1 , A_2 , and A_4), yields the CDF

$$\Pr\left(D_3 < \frac{1}{6} + v\right) = \begin{cases} 48v^3 & 0 < v < \frac{1}{6} \\ -12v^3 + 8v^2 + v - \frac{1}{9} & \frac{1}{6} < v < \frac{2}{6} \\ -4v^3 + \frac{11}{3}v - \frac{11}{27} & \frac{2}{6} < v < \frac{3}{6} \\ 2v^3 - 5v^2 + \frac{25}{6}v - \frac{17}{108} & \frac{3}{6} < v < \frac{5}{6}. \end{cases}$$

Note that Birnbaum [9] contains a sign error in the fourth v -subinterval of support.

Table 10.2 shows how rapidly the number of n -fold integrations required to compute the CDF of $D_n - \frac{1}{2n}$ grows with n . Note that the five triple integrals in

Table 10.2. Computational requirements for computing the D_n CDF for small n .

n	m	Number of n -fold integrals required for each v -subinterval	Total number of n -fold integrals required	Total number of single integrals required
1	1	1	1	1
2	2	1, 2	3	6
3	4	1, 4, 5, 3	13	39
4	5	1, 8, 13, 9, 4	35	140
5	7	1, 16, 34, 27, 28, 14, 5	125	625
6	8	1, 32, 89, 81, 89, 48, 20, 6	366	2196

the previous example corresponds to $n = 3$ and the third v -subinterval ($k = 3$), shown in boldface in the table. Note that the fifth column in Table 10.2 is the product of the first and fourth columns.

Example 10.10. The method for determining the integers in the third column of the Table 10.2 is somewhat analogous to determining the entries in a row of Pascal's triangle by using the entries of the previous row. Beginning at each nonzero A matrix element in rows 2 through n , there are a certain number of paths whose moves to the top row of the matrix satisfy the two conditions stated previously. We will assign such elements a number p (for paths) that denotes the number of paths from that element to the top of the matrix. Thus if $p_{2,2} = 2$ (as is the case for the (2, 2) element of the A_3 matrix for $n = 3$ and $k = 3$ in Example 10.9), then no matter how we reached this element of the A matrix, there are only two possible paths remaining to the top. The $p_{i,j}$ values are assigned as follows. Only A matrix elements with value 1 are assigned a $p_{i,j}$ value. If the j th entry in the first row of A is nonzero, assign $p_{1,j} = 1$. For the second and subsequent rows ($i = 2, 3, \dots, n$), $p_{i,j} = \sum_{q=1}^j p_{i-1,q}$. Upon completion, $p_{n,n}$ represents the total number of paths from the lower right corner of the matrix to the top row of the matrix. Consider, for example, the case of $n = 5$ and $k = 4$. The A_4 matrix with superscripts denoting the $p_{i,j}$ values is

$$A_4 = \begin{bmatrix} 0 & 1^1 & 1^1 & 1^1 & 0 \\ 0 & 1^1 & 1^2 & 1^3 & 1^3 \\ 0 & 0 & 1^3 & 1^6 & 1^9 \\ 0 & 0 & 0 & 1^9 & 1^{18} \\ 0 & 0 & 0 & 0 & 1^{27} \end{bmatrix}.$$

Here we see that $p_{5,5} = 27$, meaning there are 27 possible paths to the top row of A_4 , consistent with the $n = 5$ and $k = 4$ element in Table 10.2.

Phase 3: Set Limits on the Appropriate Integrals. The indicator matrix A shows, by the presence of a 1 in row i and column j , for $j < n$, that u_i can assume values from the maximum of $x_j - v$ and 0 to the minimum of $x_{j+1} - v$ and $x_i + v$. The presence of a 1 in row i and column n , means that u_i assumes values from $x_n - v$ to the minimum of $x_i + v$ and 1. Each path in A represents a set of allowable intervals of integration for the variables u_1 to u_n in a particular n -fold integral. For a particular path, the limits of integration for each u_i are those given above that correspond to entry $a_{i,j}$ of A in row i that lies on the path, with one exception: if both $a_{i,j}$ and $a_{i-1,j}$ are on the same path, the lower limit of integration for u_i must be u_{i-1} . This exception occurs because the path allows u_i and u_{i-1} to assume values in the same u -subinterval. In such a case, the lower limit of integration for u_i must be u_{i-1} in order to reflect the requirement $u_{i-1} \leq u_i$.

For each path through the matrix A , and for each nonzero entry $a_{i,j}$ on that path, a single integration with respect to u_i must be performed. If $a_{i-1,j}$ is on the same path as $a_{i,j}$, then the lower limit of integration will be the variable u_{i-1} . If $a_{i-1,j}$ is not on the same path as $a_{i,j}$, then the lower limit of integration will be a fixed number: the maximum of $x_j - v$ and 0. Thus for each path which passes through $a_{i,j}$, either a variable or fixed lower limit of integration might be appropriate.

Example 10.11. Consider again the case of $n = 5$ and the fourth v -subinterval ($k = 4$). The A_4 matrix shown below has its 1's replaced with \mathcal{F} , \mathcal{V} , or \mathcal{B} , indicating whether a fixed-limit integral, a variable-limit integral, or both need to be computed for each entry.

$$A_4 = \begin{bmatrix} 0 & \mathcal{F} & \mathcal{F} & \mathcal{F} & 0 \\ 0 & \mathcal{V} & \mathcal{B} & \mathcal{B} & \mathcal{F} \\ 0 & 0 & \mathcal{B} & \mathcal{B} & \mathcal{B} \\ 0 & 0 & 0 & \mathcal{B} & \mathcal{B} \\ 0 & 0 & 0 & 0 & \mathcal{B} \end{bmatrix}.$$

In general, when the A matrix contains a zero, neither the fixed nor variable lower limits need to be computed. Now consider the elements of the A matrix that contain a 1. The positions associated with the first 1 in each column of A require only a fixed lower limit to be calculated. The positions below the first 1 in the first nonzero column require only a variable lower limit to be calculated. All other positions in the A matrix associated with 1 elements require both a fixed and a variable lower limit to be calculated. Table 10.3 shows the computational efficiency of performing the integrations for various values of n by using the algorithm.

Table 10.3. Computational efficiency associated with using the F and V arrays.

n	m	Total number of single integrals required (Table 10.2)	Total number of single integrals required (algorithm)
1	1	1	0
2	2	6	4
3	4	39	23
4	5	140	48
5	7	625	108
6	8	2196	170
10	14	442540	800
15	22	318612735	2793
20	29	232093052440	6400
30	44	136617382366486800	21550
50	74	61056379955994182386843138150	99500

The number of single integrals calculated by the algorithm for any n is no more than

$$\frac{n(n+1)}{2} \cdot 2 \cdot \left(\left\lceil \frac{3n}{2} \right\rceil - 2 \right) \cong \frac{3}{2}n^3,$$

where the first factor corresponds to the maximum number of 1's in any A matrix, the second factor accounts for computing both the fixed and variable matrices (described subsequently), and the last factor is one less than m since no integration is required for the first v -subinterval. Due to savings associated with 0 elements in the A matrix and the fact that both fixed and variable matrix elements do not need to be calculated in all cases, this order n^3 expression is closely approximated by $0.80n^3$ for $n = 20, 30$, and 50 .

Table 10.4 lists the coefficients of the polynomials that define the CDF of $D_n - \frac{1}{2n}$ as computed by the APPL procedure KSRV whose algorithm is given at the end of this section, for $n = 1, 2, \dots, 6$. The ability to store fractions exactly is of particular benefit for these calculations.

Rather than detail every aspect of the logic of Phase 3 of the algorithm, we illustrate the evolution of the $n \times n$ matrices F (for fixed limits) and V (for variable limits) in a row-by-row fashion for a particular combination of n and k .

Example 10.12. Consider again the case of $n = 3$ and $k = 3$, which corresponds to the v -subinterval $\frac{2}{6} < v < \frac{3}{6}$. The x_i values associated with $n = 3$ are

$$x_1 = \frac{1}{6} \qquad x_2 = \frac{3}{6} \qquad x_3 = \frac{5}{6}.$$

The v_i values associated with $n = 3$ are $v_0 = 0$,

$$v_1 = \frac{1}{6} \qquad v_2 = \frac{2}{6} \qquad v_3 = \frac{3}{6} \qquad v_4 = \frac{5}{6}.$$

The centers of the v -subintervals are

$$c_1 = \frac{1}{12} \qquad c_2 = \frac{3}{12} \qquad c_3 = \frac{5}{12} \qquad c_4 = \frac{8}{12}.$$

The A_3 matrix is

$$A_3 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}.$$

The F and V matrices are computed simultaneously, row by row, starting with the last row and ending with the first. Thus all of the elements of the third row of F and of V are computed before computing the elements of the second row is begun. In general, integrals that are calculated in a particular row become integrand candidates for the integration to be carried out in the row immediately above. The F and V matrices are designed to store the intermediate results

Table 10.4. CDFs of $D_n - \frac{1}{2n}$ for $n = 1, 2, \dots, 6$.

n	m	k	Coefficients of CDF polynomials	Subinterval
1	1	1	2, 0	$0 < v < \frac{1}{2}$
2	2	1	8, 0, 0	$0 < v < \frac{1}{4}$
		2	$-2, 3, -\frac{1}{8}$	$\frac{1}{4} < v < \frac{3}{4}$
3	4	1	48, 0, 0, 0	$0 < v < \frac{1}{6}$
		2	$-12, 8, 1, -\frac{1}{9}$	$\frac{1}{6} < v < \frac{2}{6}$
		3	$-4, 0, \frac{11}{3}, -\frac{11}{27}$	$\frac{2}{6} < v < \frac{3}{6}$
		4	$2, -5, \frac{25}{6}, -\frac{17}{108}$	$\frac{3}{6} < v < \frac{5}{6}$
4	5	1	384, 0, 0, 0, 0	$0 < v < \frac{1}{8}$
		2	$-48, 0, 15, -\frac{9}{8}, \frac{3}{256}$	$\frac{1}{8} < v < \frac{2}{8}$
		3	$16, -40, 21, -\frac{5}{8}, -\frac{29}{256}$	$\frac{2}{8} < v < \frac{3}{8}$
		4	$6, -7, -\frac{27}{16}, \frac{293}{64}, -\frac{853}{2048}$	$\frac{3}{8} < v < \frac{5}{8}$
		5	$-2, 7, -\frac{147}{16}, \frac{343}{64}, -\frac{353}{2048}$	$\frac{5}{8} < v < \frac{7}{8}$
5	7	1	3840, 0, 0, 0, 0, 0	$0 < v < \frac{1}{10}$
		2	$0, -288, \frac{624}{5}, -\frac{96}{25}, -\frac{36}{125}, \frac{6}{625}$	$\frac{1}{10} < v < \frac{2}{10}$
		3	$160, -160, \frac{24}{5}, \frac{616}{25}, -\frac{332}{125}, \frac{6}{125}$	$\frac{2}{10} < v < \frac{3}{10}$
		4	$-20, 64, -\frac{318}{5}, \frac{542}{25}, \frac{343}{500}, -\frac{273}{1250}$	$\frac{3}{10} < v < \frac{4}{10}$
		5	$12, 0, -\frac{62}{5}, \frac{6}{5}, \frac{2391}{500}, -\frac{3413}{6250}$	$\frac{4}{10} < v < \frac{5}{10}$
		6	$-8, 18, -\frac{52}{5}, -\frac{19}{5}, \frac{1838}{250}, -\frac{10527}{25000}$	$\frac{5}{10} < v < \frac{7}{10}$
		7	$2, -9, \frac{81}{5}, -\frac{729}{50}, \frac{6561}{1000}, -\frac{9049}{50000}$	$\frac{7}{10} < v < \frac{9}{10}$
6	8	1	46080, 0, 0, 0, 0, 0, 0	$0 < v < \frac{1}{12}$
		2	$2880, -3360, 660, 60, -\frac{95}{12}, \frac{5}{24}, -\frac{5}{5184}$	$\frac{1}{12} < v < \frac{2}{12}$
		3	$320, 480, -700, \frac{5620}{27}, -\frac{125}{36}, -\frac{275}{216}, \frac{2195}{46656}$	$\frac{2}{12} < v < \frac{3}{12}$
		4	$-280, 420, -\frac{335}{2}, -\frac{1675}{54}, \frac{26005}{864}, -\frac{3125}{1728}, -\frac{20645}{373248}$	$\frac{3}{12} < v < \frac{4}{12}$
		5	$104, -188, \frac{1235}{6}, -\frac{7435}{54}, \frac{36245}{864}, \frac{7327}{5184}, -\frac{69797}{373248}$	$\frac{4}{12} < v < \frac{5}{12}$
		6	$-20, 22, \frac{45}{4}, -\frac{2005}{108}, -\frac{185}{1728}, \frac{57971}{10368}, -\frac{406469}{746496}$	$\frac{5}{12} < v < \frac{7}{12}$
		7	$10, -33, \frac{925}{24}, -\frac{3065}{216}, -\frac{22175}{3456}, \frac{134807}{20736}, -\frac{632863}{1492992}$	$\frac{7}{12} < v < \frac{9}{12}$
		8	$-2, 11, -\frac{605}{24}, \frac{6655}{216}, -\frac{73205}{3456}, \frac{161051}{20736}, -\frac{278569}{1492992}$	$\frac{9}{12} < v < \frac{11}{12}$

of integration of the n -fold integrals so that the necessary inner integration operations are performed only once. Entries in the i th rows of F and V each result from $n + 1 - i$ single integrations, one integration corresponding to the i th row and one integration for every lower row. The F matrix stores the results of all integration in which the last integral had a fixed lower limit and the V matrix stores the results of all integration in which the last integral had a variable lower limit. In

this example, for the inner-most integration variable u_3 , there is only one possible fixed integration to compute and one possible variable integration to compute on the third u -subinterval. These computations are seen in the (3, 3) element of the respective matrices:

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & F_{1,3} \\ 0 & F_{2,2} & F_{2,3} \\ 0 & 0 & \int_{\frac{5}{6}-v}^1 1 du_3 \end{bmatrix}, \quad V = \begin{bmatrix} V_{1,1} & V_{1,2} & V_{1,3} \\ 0 & V_{2,2} & V_{2,3} \\ 0 & 0 & \int_{u_2}^1 1 du_3 \end{bmatrix}.$$

The second variable, u_2 , has four possible combinations of limits and integrands. Thus there are four storage elements for u_2 , found in the second rows of the F and V matrices. Elements (2, 2) and (2, 3) of the F matrix store integration results that have all fixed lower-limits of integration, but the integrands are fixed and variable, respectively. Similarly, elements (2, 2) and (2, 3) of the V matrix are for the cases when the lower-limit of integration contains the u_1 variable, while the integrands are fixed and variable, respectively. $F_{2,2}$ is computed to cover the case in which u_2 varies over the second u -subinterval and u_3 varies over the third u -subinterval. $F_{2,3}$ covers the case in which both u_2 and u_3 vary over the third u -subinterval, so u_3 must have u_2 as its variable lower limit; hence the integrand is $V_{3,3}$, not $F_{3,3}$. Similar cases are covered by $V_{2,2}$ and $V_{2,3}$, except that variable lower limits are used in anticipation of the fact that u_1 and u_2 can both vary over the same u -subinterval, making it necessary to have u_1 as the (variable) lower limit for u_2 . The integrations in the second row of each matrix are shown below (note, the integration for the (3, 3) element, previously discussed, has been carried out):

$$F = \begin{bmatrix} F_{1,1} & F_{1,2} & F_{1,3} \\ 0 & \int_{\frac{1}{2}-v}^{\frac{5}{6}-v} F_{3,3} du_2 & \int_{\frac{5}{6}-v}^{\frac{1}{2}+v} V_{3,3} du_2 \\ 0 & 0 & \frac{1}{6} + v \end{bmatrix},$$

$$V = \begin{bmatrix} V_{1,1} & V_{1,2} & V_{1,3} \\ 0 & \int_{u_1}^{\frac{5}{6}-v} F_{3,3} du_2 & \int_{u_1}^{\frac{1}{2}+v} V_{3,3} du_2 \\ 0 & 0 & 1 - u_2 \end{bmatrix}.$$

For the third and last variable, u_1 , only fixed-limit integration takes place so only the F matrix gets updated. The (1, 1) element covers the case in which u_1 varies over the first u -subinterval and u_2 varies over the second and third u -subinterval. The (1, 2) element covers the case in which u_1 varies over the second u -subinterval and u_2 varies over the second and third u -subinterval. The (1, 3) element allows both u_1 and u_2 to vary over the third u -subinterval. The integrals in this first row are shown below (note again, all previously discussed integration has been carried out in the second and third rows of F and V):

$$F = \begin{bmatrix} \int_0^{\frac{1}{2}-v} (F_{2,2} + F_{2,3}) du_1 & \int_{\frac{1}{2}-v}^{\frac{5}{6}-v} (V_{2,2} + F_{2,3}) du_1 & \int_{\frac{5}{6}-v}^{\frac{1}{6}+v} V_{2,3} du_1 \\ 0 & \frac{1}{3}v + \frac{1}{18} & -\frac{1}{9} + \frac{2}{3}v \\ 0 & 0 & \frac{1}{6} + v \end{bmatrix},$$

$$V = \begin{bmatrix} V_{1,1} & V_{1,2} & V_{1,3} \\ 0 & -v^2 + \frac{2}{3}v + \frac{5}{36} - vu_1 - \frac{1}{6}u_1 & \frac{1}{2}v^2 + \frac{1}{2}v + \frac{3}{8} + \frac{1}{2}u_1^2 - u_1 \\ 0 & 0 & 1 - u_2 \end{bmatrix}.$$

The completely evaluated F matrix is given by

$$F = \begin{bmatrix} -v^2 + \frac{5}{9}v - \frac{1}{36} & -\frac{1}{36} + \frac{5}{18}v & -\frac{2}{3}v^3 + v^2 - \frac{2}{9}v - \frac{1}{81} \\ 0 & \frac{1}{3}v + \frac{1}{18} & -\frac{1}{9} + \frac{2}{3}v \\ 0 & 0 & \frac{1}{6} + v \end{bmatrix}.$$

Finally, the CDF on the 3rd v -subinterval is the sum of the elements in the first row of the F matrix, all multiplied by $3! = 6$.

Phase 4: Shift the Distribution. At this point, the CDF is computed in the form $\Pr(D_n < v + \frac{1}{2n})$. Now we convert the distribution into the more usable form $\Pr(D_n < y) = F_{D_n}(y)$ by making the substitution $y = v + \frac{1}{2n}$ in both the polynomials and the v -subintervals of the CDF. Specifically, we add $\frac{1}{2n}$ to each endpoint of the v -subintervals and we substitute $(y - \frac{1}{2n})$ for v in the CDF polynomials. We then take these two lists of numbers and polynomials respectively, simplify them (using Maple's `simplify()` command), and create the CDF representation in the usual list-of-sublists representation for distributions. Specifically, we can now verify critical values for the distribution, but more importantly, we can calculate exact significance levels of any given statistic. An example of the shifted distribution for $n = 6$ is

$$F(y) = \begin{cases} 0 & y < \frac{1}{12} \\ 46080y^6 - 23040y^5 + 4800y^4 - \frac{1600}{3}y^3 + \frac{100}{3}y^2 - \frac{10}{9}y + \frac{5}{324} & \frac{1}{12} \leq y < \frac{1}{6} \\ 2880y^6 - 4800y^5 + 2360y^4 - \frac{1280}{3}y^3 + \frac{235}{9}y^2 + \frac{10}{27}y - \frac{5}{81} & \frac{1}{6} \leq y < \frac{1}{4} \\ 320y^6 + 320y^5 - \frac{2600}{3}y^4 + \frac{4240}{9}y^3 - \frac{785}{9}y^2 + \frac{145}{27}y - \frac{35}{1296} & \frac{1}{4} \leq y < \frac{1}{3} \\ -280y^6 + 560y^5 - \frac{1115}{3}y^4 + \frac{515}{9}y^3 + \frac{1525}{54}y^2 - \frac{565}{81}y + \frac{5}{16} & \frac{1}{3} \leq y < \frac{5}{12} \\ 104y^6 - 240y^5 + 295y^4 - \frac{1985}{9}y^3 + \frac{775}{9}y^2 - \frac{7645}{648}y + \frac{5}{16} & \frac{5}{12} \leq y < \frac{1}{2} \\ -20y^6 + 32y^5 - \frac{185}{9}y^3 + \frac{175}{36}y^2 + \frac{3371}{648}y - 1 & \frac{1}{2} \leq y < \frac{2}{3} \\ 10y^6 - 38y^5 + \frac{160}{3}y^4 - \frac{265}{9}y^3 - \frac{115}{108}y^2 + \frac{4651}{648}y - 1 & \frac{2}{3} \leq y < \frac{5}{6} \\ -2y^6 + 12y^5 - 30y^4 + 40y^3 - 30y^2 + 12y - 1 & \frac{5}{6} \leq y < 1 \\ 1 & y \geq 1. \end{cases}$$

The CDF and PDF of D_6 are shown in Figures 10.8 and 10.9. The APPL commands used to produce these plots are

```
> Y := KSRV(6);
> PlotDist(CDF(Y));
> PlotDist(PDF(Y));
```

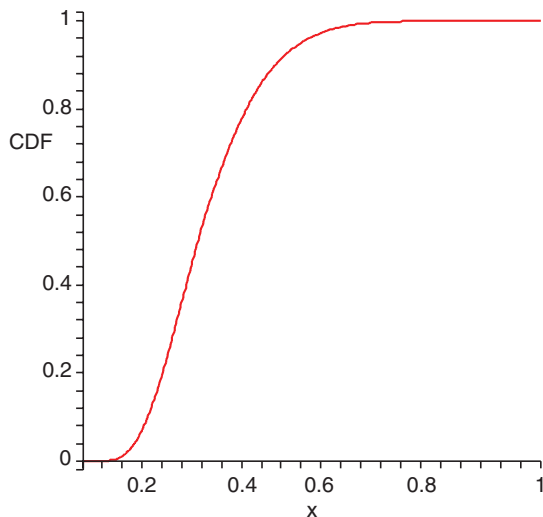


Fig. 10.8. The CDF of the D_6 random variable.

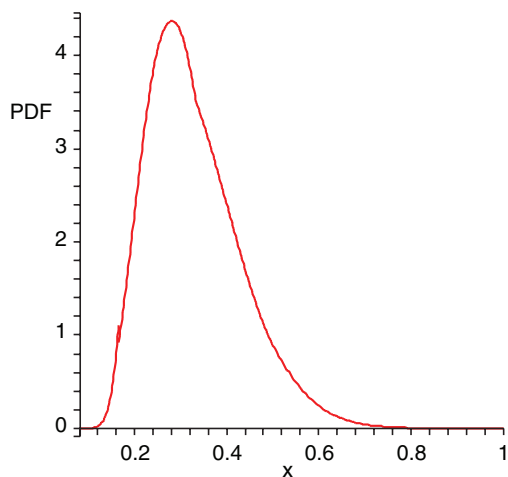


Fig. 10.9. The PDF of the D_6 random variable. Note the discontinuity at $y = 1/6$.

We have noted that there is a discontinuity in the PDF at $y = \frac{1}{6}$, and more generally, at $y = \frac{1}{n}$. This corresponds to the value $v = \frac{1}{2n}$ in Birnbaum's original integral. This is the smallest value of v for which the n -dimensional hypercube centered at $(\frac{1}{2n}, \frac{3}{2n}, \dots, \frac{2n-1}{2n})$ makes contact with the boundary of the region in n -space that satisfies $0 \leq u_1 \leq u_2 \leq \dots \leq u_n \leq 1$.

Critical values associated with the distribution of D_n are determined with the APPL IDF procedure. Arbitrary accuracy can be achieved by using the Maple `Digits` command to define the precision. We have verified critical

values by comparing them with those given in Owen [66] for $\alpha = 0.1, 0.05$, and 0.01 and for $n = 1, 2, \dots, 10$ and those values match ours exactly, except that ours are not limited to five decimal places.

In summary, we have developed and implemented an algorithm for computing the CDF of the KS test statistic in the all-parameters-known case for a positive integer parameter n . Computing the CDF provides a challenging calculus problem, even for small values of n . The CDF can be constructed from a set of piecewise degree- n polynomials. The algorithm can be used to plot the entire CDF or to just find particular fractiles or probabilities associated with the distribution. Further details are given in Drew et al. [17]. An extension to the parameters-estimated-from-data case for sampling from an exponential population and a small sample size is given in Evans et al. [21]. We conclude this section by presenting the algorithm for KSRV, which returns the CDF of the KS test statistic D_n in the all-parameters-known case.

Procedure KSRV: Compute the CDF of the Kolmogorov–Smirnov test statistic.

Input: A positive integer n , the sample size associated with the KS test statistic when all parameters are known.

Output: The CDF of the KS test statistic when all parameters are known is returned. The random variable will be in the usual list-of-sublists format.

Algorithm:

Phase 1: Determine the endpoints of the segments that define the support of the KS test statistic.

Dimension $v \left[0.. \left\lceil \frac{3n}{2} \right\rceil - 1 \right]$ v holds the partition values for the support of D_n
 $m \leftarrow 0$
 $v_m \leftarrow 0$
 $g \leftarrow \frac{1}{2n}$ g is the gap between adjacent integration limits when $v = 0$
 For $i \leftarrow 1$ to $n - 1$ by 1
 $m \leftarrow m + 1$
 $v_m \leftarrow i \cdot g$
 For $j \leftarrow 2 \cdot \left\lfloor \frac{n}{2} \right\rfloor + 1$ to $2n - 1$ by 2 lower limit is n or $n + 1$, whichever is odd
 $m \leftarrow m + 1$
 $v_m \leftarrow j \cdot g$

Phase 2: This phase of the algorithm has three parts to it.

1. Define c_1, c_2, \dots, c_m as the midpoints of the support intervals
2. Define x_1, x_2, \dots, x_n as the limits of integration when $v = 0$
3. Define the $n \times n$ indicator matrices A_1, A_2, \dots, A_m .

Dimension $c[m]$ define vector c of midpoints of the support subintervals
 For $k \leftarrow 1$ to m
 $c_k \leftarrow (v_{k-1} + v_k)/2$

Dimension $x[n]$ define vector x of centers of integration limits

For $i \leftarrow 1$ to n

$x_i \leftarrow (2i - 1) \cdot g$

Dimension $A[n, n]$

For $i \leftarrow 2$ to n

For $j \leftarrow 1$ to $i - 1$

$A_{ij} \leftarrow 0$

zero out lower triangle of A

For $k \leftarrow 1$ to m

loop through all subintervals of support

For $i \leftarrow 1$ to n

For $j \leftarrow i$ to n

$A_{ij} \leftarrow 0$

zero out upper triangle of A

$z \leftarrow \max\{\lfloor nc_k - \frac{1}{2} \rfloor, 0\}$

$z \equiv$ number of leading zero columns in A

$l \leftarrow \min\{\lceil 2nc_k \rceil, n\}$ $l \equiv$ number of u -subint. intersecting $(\frac{1}{2n} - v, \frac{1}{2n} + v)$

For $i \leftarrow 1$ to n

For $j \leftarrow \max\{i, z + 1\}$ to $\min\{n, i + l - 1\}$

$A_{ij} \leftarrow 1$

place 1's in A

Phase 3: Compute the fixed and variable limit matrices F and V and combine to give the piecewise polynomial CDF.

Dimension $P[m]$ n th order polynomials for CDF segments param. by v

Dimension $F[n, n]$ array of fixed integrals parameterized by v

Dimension $V[n, n]$ array of variable integrals parameterized by v

Real $S(v)$ a function of v , used as a running sum $S(v)$ is the sum of all F elements one row down, starting one element to the right

$P_1(v) \leftarrow n!(2v)^n$ $\left[P_1(v) \leftarrow n! \int_{x_1-v}^{x_1+v} \int_{x_2-v}^{x_2+v} \dots \int_{x_n-v}^{x_n+v} 1 du_n \dots du_2 du_1 \right]$

For $k \leftarrow 2$ to m

loop through all subintervals of support

$z \leftarrow \max\{\lfloor nc_k - \frac{1}{2} \rfloor, 0\}$

$l \leftarrow \min\{\lceil 2nc_k \rceil, n\}$

$F_{n,n}(v) \leftarrow \int_{x_n-v}^1 1 du_n$

$V_{n,n}(v) \leftarrow \int_{u_{n-1}}^1 1 du_n$

For $i \leftarrow n - 1$ to 2 by -1

If $i + l > n$ then

$S(v) \leftarrow 0$

else

$S(v) \leftarrow F_{i+1,i+l}(v)$

If $i + l > n + 1$ then

$F_{i,n}(v) \leftarrow \int_{x_n-v}^{\lfloor x_i+c_k \rfloor + (x_i+v)(1-\lfloor x_i+c_k \rfloor)} V_{i+1,n}(v) du_i$

$V_{i,n}(v) \leftarrow \int_{u_{i-1}}^{\lfloor x_i+c_k \rfloor + (x_i+v)(1-\lfloor x_i+c_k \rfloor)} V_{i+1,n}(v) du_i$

If $i + l = n + 1$ then
 $F_{i,n}(v) \leftarrow \int_{x_n-v}^{x_i+v} V_{i+1,n}(v) du_i$
 If $i + l < n + 1$ then
 $F_{i,i+l-1}(v) \leftarrow \int_{x_{i+l-1}-v}^{x_i+v} \{V_{i+1,i+l-1}(v) + S(v)\} du_i$
 $S(v) \leftarrow S(v) + F_{i+1,\min\{i+l-1,n\}}(v)$
 For $j \leftarrow \min\{n-1, i+l-2\}$ to $\max\{i+1, z+2\}$ by -1
 $F_{i,j}(v) \leftarrow \int_{x_j-v}^{x_{j+1}-v} \{V_{i+1,j}(v) + S(v)\} du_i$
 $V_{i,j}(v) \leftarrow \int_{u_{i-1}}^{x_{j+1}-v} \{V_{i+1,j}(v) + S(v)\} du_i$
 $S(v) \leftarrow S(v) + F_{i+1,j}(v)$
 If $z + 1 \leq i$ then
 $V_{i,i}(v) \leftarrow \int_{u_{i-1}}^{x_{i+1}-v} S(v) du_i$
 If $z + 1 > i$ then
 $V_{i,z+1}(v) \leftarrow \int_{u_{i-1}}^{x_{z+2}-v} \{V_{i+1,z+1}(v) + S(v)\} du_i$
 If $z + 1 < i$ then
 $F_{i,i}(v) \leftarrow \int_{x_i-v}^{x_{i+1}-v} S(v) du_i$
 If $l = n$ then rows $2, 3, \dots, n$ of A matrix completed at this point
 $S(v) \leftarrow 0$
 $F_{1,n}(v) \leftarrow \int_{x_n-v}^{x_1+v} V_{2,n}(v) du_1$
 else
 $S(v) \leftarrow F_{2,l+1}(v)$
 If $l < n$ then
 $F_{1,l}(v) \leftarrow \int_{x_l-v}^{x_1+v} \{V_{2,l}(v) + S(v)\} du_1$
 $S(v) \leftarrow S(v) + F_{2,l}(v)$
 For $j \leftarrow \min\{n-1, l-1\}$ to $\max\{2, z+1\}$ by -1
 $F_{1,j}(v) \leftarrow \int_{(x_j-v)\lceil x_j-c_k \rceil}^{x_{j+1}-v} \{V_{2,j}(v) + S(v)\} du_1$
 $S(v) \leftarrow S(v) + F_{2,j}(v)$
 If $z = 0$ then
 $F_{1,1}(v) \leftarrow \int_0^{x_2-v} S(v) du_1$
 $P_k(v) \leftarrow 0$
 For $j \leftarrow z+1$ to l
 $P_k(v) \leftarrow P_k(v) + F_{1,j}(v)$
 $P_k(v) \leftarrow n!P_k(v)$

Phase 4: Shift the distribution to the form of $\Pr(D_n < y)$ by substituting $y = v + \frac{1}{2n}$.

$y_0 \leftarrow v_0 + \frac{1}{2n}$
 For $k \leftarrow 1$ to m loop through all subintervals of support
 $y_i \leftarrow v_k + \frac{1}{2n}$
 $F_k(y) \leftarrow P_k(y - \frac{1}{2n})$

Return: y_0, y_1, \dots, y_m and $F_1(y), F_2(y), \dots, F_m(y)$.

Other Applications

This chapter contains miscellaneous computational probability applications. Section 11.1 concerns algorithms for calculating the probability distribution of the longest path of a series-parallel stochastic activity network with continuous activity durations. Section 11.2 concerns the use of APPL in determining whether a continuous random variable obeys Benford's law. Finally, Section 11.3 contains miscellaneous computational probability applications that are not covered elsewhere in the monograph.

11.1 Stochastic Activity Networks

Project planning is typically aided by the use of *activity networks*, which show precedence relationships between the various activities that constitute the project [1]. An activity network is a special case of a directed graph in which the nodes (or vertices) represent points in time and the arcs (or edges) represent activities with time values modeling activity durations. Figure 11.1 shows an example of a *stochastic* activity network, where the positive random variable Y_{ij} denotes the time to complete the activity associated with the arc from node i to node j . The Y_{ij} 's are assumed to be mutually independent. Common-cause delays, e.g., rain delays on an outdoor project, are not considered here. Activity start times are constrained in that no activity emanating from a given node can start until all activities that enter that node have been completed.

Notation and Assumptions. Activity networks have n nodes and m arcs, with a single source node (labeled node 1), a single terminal node (labeled node n), and no loops. Nodes are labeled $1, 2, \dots, n$ in topological order as defined by Laplante [45] so that if there is an arc a_{ij} from node i to node j , then i is less than j . Thus the nodes along each path occur in ascending order.

For each node j , the “backward” set $\mathcal{B}(j)$ is the set of all nodes that are immediately *before* node j on some path, that is, $\mathcal{B}(j)$ is the set of all nodes i

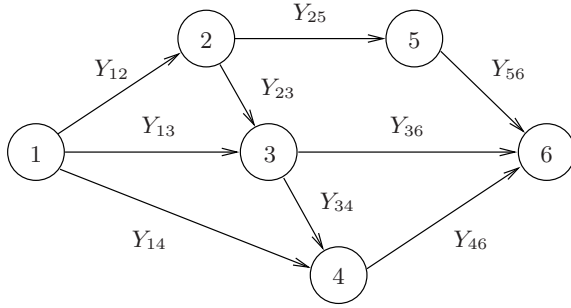


Fig. 11.1. A six-node, nine-arc stochastic activity network.

for which there is an arc from i to j . Similarly, for each node j , the “forward” set $\mathcal{A}(j)$ is the set of all nodes that are immediately *after* node j on some path, that is, $\mathcal{A}(j)$ is the set of all nodes k for which there is an arc from j to k .

For each arc a_{ij} , there is a continuous random activity duration Y_{ij} with positive support. The distribution of Y_{ij} is represented by its CDF $F_{Y_{ij}}(t)$ or its PDF $f_{Y_{ij}}(t)$. For each node j there is a random time value T_j , which is the time of completion of all activities entering node j . The quantity T_n is therefore the time of completion of the entire network. Shier [76, pages 122–123] gives an example of an activity network with discrete activity durations.

For each network, there will typically be several paths leading from node 1 to node n . Let M be the set of all paths, with the paths labeled $\pi_1, \pi_2, \dots, \pi_r$, where $r = |M|$ is the number of paths. A path may be viewed as an ordered set of arcs leading in succession from node 1 to node n . The *length* of path π_k , denoted L_k , is the sum of all Y_{ij} corresponding to the arcs $a_{ij} \in \pi_k$.

For each realization of a stochastic network, the *critical path* π_c is the path with the greatest length, $L_c = \max\{L_1, L_2, \dots, L_r\}$. The length of the critical path determines the time to complete the entire network. For a stochastic network, a path in M is the critical path with some probability $p(\pi_k) = \Pr(L_k = L_c)$, $k = 1, 2, \dots, r$. Some arcs may be along more than one path. The probability that arc a_{ij} is along the critical path, also called the arc’s *criticality*, denoted by ρ_{ij} , is the sum of all $p(\pi_k)$ where $a_{ij} \in \pi_k$. The criticality of arc a_{ij} can be calculated by

$$\rho_{ij} = \sum_{k=1}^r p(\pi_k) \delta_{ij}^k,$$

where δ_{ij}^k is 1 if $a_{ij} \in \pi_k$ and 0 otherwise.

Matrix Representation of a Network. The first step in building a model is defining a mathematical representation of a network. Matrices are well suited for this task because (a) each node and arc in the network can be

designated by using the rows and columns of a matrix, and (b) matrices are easily instantiated in most computer languages. A node–arc incidence matrix was chosen due to its suitability for the algorithms developed here.

The node–arc incidence matrix associated with a network is an $n \times m$ matrix N , where each row represents a node and each column represents an arc. Let

$$N[i, k] = \begin{cases} 1 & \text{if arc } k \text{ leaves node } i \\ -1 & \text{if arc } k \text{ enters node } i \\ 0 & \text{otherwise.} \end{cases}$$

A node–arc incidence matrix that describes the network in Figure 11.1 is

$$N = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}.$$

The only limitation of this representation is that it cannot show an arc that leaves and enters the same node. This ability will not be necessary since feedback is not allowed by our definition of a stochastic activity network.

Series–Parallel Networks. The traditional approach to analyzing stochastic activity networks is simulation. While straightforward to apply, simulation has a distinct drawback. Each additional digit of accuracy in the estimate of $E[T_n]$, for example, requires approximately a 100-fold increase in replications due to the square root in the denominator of the confidence interval formulas for $E[T_n]$. We now derive the various performance measures analytically—eliminating the need for simulation—for a special case of a stochastic activity network known as a *series–parallel network*.

A series–parallel activity network is a special case of an activity network that can be simplified by a sequence of reductions to a simple network consisting of just two nodes and one arc. Reductions consist of taking two arcs, either in series or in parallel, and replacing them with a single arc that has a random duration whose distribution is calculated in the algorithm. This is a good match for APPL’s symbolic capabilities because each reduction can be preformed using either the **Maximum** or **Convolution** procedures. Once the network is completely reduced, the distribution of the single remaining arc is the distribution of the time T_n to complete the original network. The recursive algorithm then reconstructs the network to determine the critical path probabilities and arc criticalities. Our algorithm will use these operations to simplify a series–parallel network, and then, by determining the distribution of T_n and reconstructing the network, determine the critical path probabilities and the criticalities. The earliest work on the exact analysis of series–parallel networks is Martin [59]. The next paragraphs describe the parallel and series reductions and reconstructions.

Parallel Reduction. The algorithm described subsequently is capable of processing two arcs in *parallel*, as illustrated in Figure 11.2. A parallel reduction is the process of combining these two arcs into a single arc. Let X_{ij} and Y_{ij} denote the random arc durations. Without loss of generality, if $T_i = 0$, then $T_j = \max\{X_{ij}, Y_{ij}\}$. So

$$F_{T_j}(t) = \Pr[T_j \leq t] = \Pr[X_{ij} \leq t \text{ and } Y_{ij} \leq t] = F_{X_{ij}}(t)F_{Y_{ij}}(t)$$

on the support of T_j . The portion of the algorithm prior to the second recursive call uses this formula to reduce two arcs in parallel to a single arc.

Parallel Reconstruction. Consider the single arc a_{ij} resulting from a parallel reduction. If the probability ρ_{ij} that this arc is on the critical path is known, then this probability can be allocated to the two original parallel arcs x and y based on each arc's activity duration CDF:

$$\rho_x = \Pr[X_{ij} > Y_{ij}] \cdot \rho_{ij} = \Pr[X_{ij} - Y_{ij} > 0] \cdot \rho_{ij}$$

and

$$\rho_y = \rho_{ij} - \rho_x.$$

This calculation involves determining the distribution of the difference between the two random variables, which can be calculated using the APPL **Difference** procedure. The portion of the algorithm after the second recursive call calculates the distribution of $X_{ij} - Y_{ij}$ and the probability that this random variable is positive.

Series Reduction. The algorithm finds the CDF of the time to complete two arcs in *series*, as illustrated in Figure 11.3. Let Y_{ij} and Y_{jk} denote the durations of the two arcs with CDFs $F_{Y_{ij}}(t)$ and $F_{Y_{jk}}(t)$. Without loss of generality, if $T_i = 0$, then $T_k = Y_{ij} + Y_{jk}$. The CDF of T_k is [12, page 215]

$$F_{T_k}(t) = \Pr[T_k \leq t] = \int_0^t F_{Y_{ij}}(t - y_{jk}) f_{Y_{jk}}(y_{jk}) dy_{jk}.$$

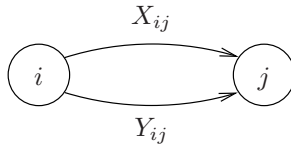


Fig. 11.2. Two arcs in parallel.

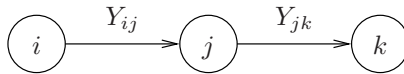


Fig. 11.3. Two arcs in series.

Series Reconstruction. Consider the single arc a_{ik} resulting from a series reduction. For any two arcs a_{ij} and a_{jk} that have been reduced into a single arc a_{ik} , $\rho_{ij} = \rho_{jk} = \rho_{ik}$. If two arcs in series are along the critical path, then so is the respective arc resulting from the reduction.

Algorithm. Two arcs in series are detected by searching the rows of the node–arc incidence matrix N for the first row containing exactly two nonzero elements whose sum is 0. Such a row represents a single arc entering and a single arc exiting the given node. To reduce a series of two arcs to a single arc, the algorithm

- zeros out the row of N corresponding to the deleted middle node,
- links the first node of the series directly to the last node of the series, and
- computes the distribution of the sum of the durations of the two arcs.

Two arcs in parallel are identified by two identical nonzero columns of N , which represent two arcs linking the same two nodes. The algorithm iterates through each column and if parallel arcs are found, the algorithm

- computes the distribution of the maximum of the two arc durations, and
- zeros out the column of N corresponding to the eliminated arc.

The following recursive algorithm reduces a series–parallel network, where N is the node–arc incidence matrix of the network. The PDFs for the activity durations of each arc are stored in a vector f (indexed in the same way that the arcs are indexed to correspond to the columns of the matrix N), and the criticalities of each arc are stored in a vector C . This algorithm recursively performs one arc reduction at a time, arbitrarily giving priority to series reductions over parallel reductions, until the network consists of just two nodes connected by a single arc. At this point, the value of C for that one arc is set to one and then the network is reconstructed on the return calls in reverse order, determining values of C for each pair of series or parallel arcs based on the value of C for the reduced arc. The algorithm returns the distribution of the time to complete the network and the final value of C , which contains ρ_{ij} for each arc a_{ij} in the network.

Procedure GetCriticalities: Find the distribution of the time to complete a stochastic activity network and the arc criticalities.

Input: The $n \times m$ node–arc incidence matrix N , the PDFs of m activity durations Y_{ij} stored in the vector f , the number of nodes n , the number of arcs m , the number of arcs to be reduced *arcsremaining* (initialized to m).

Output: Criticalities $C[m]$

Algorithm:

local $C, i, j, \text{rowsum}, \text{absrowsum}, \text{negidx}, \text{posidx}, c, d, l, k, \text{empty}, \text{numdiff}$
 if (*arcsremaining* = 1) if network reduced to one arc

```

for  $i \leftarrow 1$  to  $n$                                 loop through rows of  $N$ 
  for  $k \leftarrow 1$  to  $m$                             loop through columns of  $N$ 
    if  $(N[i, k] \neq 0)$                             if true then arc  $k$  is the last existing arc
       $C[k] \leftarrow 1$                                 set criticality to 1
      print(CDF( $f[k]$ ))                            print CDF of project completion time
      return( $C$ )                                    return criticalities
for  $i \leftarrow 1$  to  $n$                                 begin series reduction
  rowsum  $\leftarrow 0$                                 initialize rowsum
  absrowsum  $\leftarrow 0$                             initialize sum of absolute values in a row
  for  $k \leftarrow 1$  to  $m$                             loop through columns of  $N$ 
    rowsum  $\leftarrow$  rowsum +  $N[i, k]$                 sum values in the  $i$ th row of  $N$ 
    absrowsum  $\leftarrow$  absrowsum +  $|N[i, k]|$         sum the absolute values
    if  $(N[i, k] = -1)$                             if arc  $k$  enters node  $i$ 
      negidx  $\leftarrow k$                             store the index of arc  $k$ 
    if  $(N[i, k] = 1)$                             if arc  $k$  leaves from node  $i$ 
      posidx  $\leftarrow k$                             store the index of arc  $k$ 
  if (rowsum = 0 and absrowsum = 2) only one arc enters & leaves node  $i$ 
     $c \leftarrow f[negidx]$                             save the PDF of first arc for reconstruction
     $d \leftarrow f[posidx]$                             save the PDF of second arc for reconstruction
     $f[negidx] \leftarrow \text{Convolution}(c, d)$         combine the arcs into a single arc
  for  $j \leftarrow 1$  to  $n$                             loop through rows of  $N$ 
     $N[j, negidx] \leftarrow N[j, negidx] + N[j, posidx]$  change entering node
     $N[j, posidx] \leftarrow 0$                         delete middle node
    arcsremaining  $\leftarrow$  arcsremaining - 1        reduce # of arcs in network
     $C \leftarrow \text{GetCriticalities}(N, f, n, m, \text{arcsremaining})$  recursive call
  for  $j \leftarrow 1$  to  $n$                                 begin series reconstruction
    if  $(N[j, negidx] = -1)$                             if arc  $negidx$  enters node  $j$ 
       $N[j, negidx] \leftarrow 0$                         delete arc made previously by reduction
       $N[j, posidx] \leftarrow -1$                     delete arc made previously by reduction
     $N[i, negidx] \leftarrow -1$                     reconstruct node deleted previously by reduction
     $N[i, posidx] \leftarrow 1$                     reconstruct node deleted previously by reduction
     $f[negidx] \leftarrow c$                             reconstruct arc duration
     $f[posidx] \leftarrow d$                             reconstruct arc duration
     $C[posidx] \leftarrow C[negidx]$                 reconstruct arc criticality
    return( $C$ )                                    return criticalities
for  $l \leftarrow 1$  to  $m$                                 begin parallel reduction
  for  $k \leftarrow l + 1$  to  $m$                             loop through columns of  $N$ 
    empty  $\leftarrow 0$                                 initialize empty
    numdiff  $\leftarrow 0$                             initialize numdiff
    for  $i \leftarrow 1$  to  $n$                             loop through rows of  $N$ 
      if  $(N[i, l] \neq N[i, k])$                     if column difference found
        numdiff  $\leftarrow 1$                         set boolean variable numdiff
      if  $(N[i, l] \neq 0)$                             if a column is not empty
        empty  $\leftarrow 1$                             set boolean variable empty
    if (numdiff = 0 and empty  $\neq 0$ )                cols  $l$  &  $k$  identical & nonzero

```

```

 $c \leftarrow f[l]$                                 save the PDF of arc  $l$  for reconstruction
 $d \leftarrow f[k]$                                 save the PDF of arc  $k$  for reconstruction
 $f[l] \leftarrow \text{Maximum}(c, d)$                 store PDF of the maximum
for  $i \leftarrow 1$  to  $n$                             loop through rows of  $N$ 
     $N[i, k] \leftarrow 0$                         zero out column of removed arc
     $\text{arcsremaining} \leftarrow \text{arcsremaining} - 1$     reduce # network arcs
     $C \leftarrow \text{GetCriticalities}(N, f, n, m, \text{arcsremaining})$  recursive call
    for  $i \leftarrow 1$  to  $n$                         begin parallel reconstruction
         $N[i, k] \leftarrow N[i, l]$                 reinsert arc  $k$  by copying column  $l$ 
         $f[l] \leftarrow c$                         rebuild distribution
         $f[k] \leftarrow d$                         rebuild distribution
         $C[k] \leftarrow \text{CDF}(\text{Difference}(f[l], f[k]), 0) \cdot C[l]$   $\text{Pr}[X_{il} - Y_{il} > 0] \cdot \rho_{il}$ 
         $C[l] \leftarrow C[l] - C[k]$                 calculate  $\rho_y = \rho_{il} - \rho_x$ 
    return( $C$ )                                    return criticalities
return( $C$ )                                        return criticalities

```

This algorithm requires symbolic processing capability in order to calculate the distribution of the maximum of two independent random variables for parallel reduction and the distribution of the sum of two independent random variables for series reduction. The APPL procedures **Maximum** and **Convolution** are used for these operations.

Example 11.1. Figure 11.4 shows an example of a series-parallel network from Elmaghraby [20, page 261]. This network can be described by the node-arc incidence matrix

$$N = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}.$$

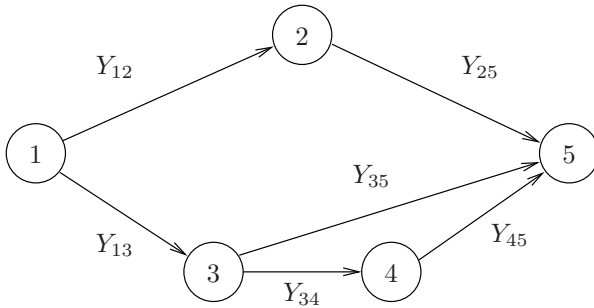


Fig. 11.4. Series-parallel network from Elmaghraby [20, page 261].

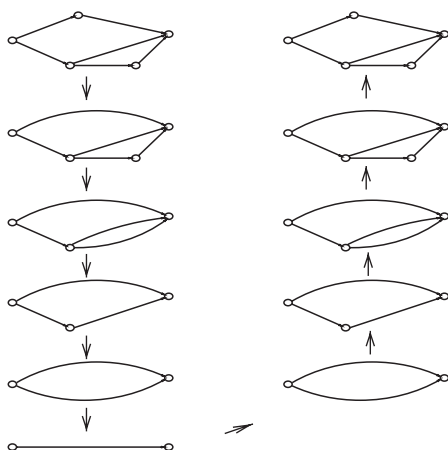


Fig. 11.5. Reduction and reconstruction of the series-parallel network from Elmaghraby [20, page 261].

Table 11.1. Paths π_k and critical path probabilities $p(\pi_k)$ for exponentially distributed arc durations with $b = 2$.

k	Node sequence	π_k	$p(\pi_k)$
1	$1 \rightarrow 2 \rightarrow 5$	$\{a_{12}, a_{25}\}$	$115/432 \cong 0.266$
2	$1 \rightarrow 3 \rightarrow 5$	$\{a_{13}, a_{35}\}$	$317/1728 \cong 0.183$
3	$1 \rightarrow 3 \rightarrow 4 \rightarrow 5$	$\{a_{13}, a_{34}, a_{45}\}$	$317/576 \cong 0.550$

The network can be reduced and reconstructed as illustrated in the steps separated by arrows in Figure 11.5. The algorithm has been applied to this stochastic activity network with the duration of each arc $Y_{ij} \sim \text{exponential}(b)$, where $1/b$ is the mean activity duration. The CDF of the time to complete the network (T_5) is returned by the algorithm as

$$F_{T_5}(t) = 1 - 3bte^{-bt} - \frac{b^2t^2}{2}e^{-bt} - 3e^{-2bt} + \frac{5b^2t^2}{2}e^{-2bt} + \frac{b^3t^3}{2}e^{-2bt} + 2e^{-3bt} + 3bte^{-3bt} + b^2t^2e^{-3bt} \quad t > 0.$$

The $r = 3$ paths in the network are listed in Table 11.1, along with the critical path probabilities. Table 11.2 contains the criticalities for the $m = 6$ arcs when $b = 2$. (Fisher, et al. [25] determine criticalities when activity durations are iid gamma random variables.)

This example has a small number of nodes and arcs along with simple distributions for the arc durations so as to make the final CDF for the project duration simple. If, however, the arc durations are each

Table 11.2. Criticalities ρ_{ij} for the exponentially distributed arc durations with $b = 2$.

Arc	Paths	ρ_{ij}
a_{12}	π_1	$115/432 \cong 0.266$
a_{13}	π_2, π_3	$317/432 \cong 0.734$
a_{25}	π_1	$115/432 \cong 0.266$
a_{35}	π_2	$317/1728 \cong 0.183$
a_{34}	π_3	$317/576 \cong 0.550$
a_{45}	π_3	$317/576 \cong 0.550$

Table 11.3. Criticalities ρ_{ij} for triangular(0, 1, 2) arc durations.

Arc	Paths	ρ_{ij}
a_{12}	π_1	$30233/237600 \cong 0.127$
a_{13}	π_2, π_3	$207367/237600 \cong 0.873$
a_{25}	π_1	$30233/237600 \cong 0.127$
a_{35}	π_2	$6013643/85536000 \cong 0.070$
a_{34}	π_3	$68638477/85536000 \cong 0.802$
a_{45}	π_3	$68638477/85536000 \cong 0.802$

independent triangular(0, 1, 2) random variables, then the problem of determining the distribution of T_5 becomes much more tedious. The arc criticalities returned by the algorithm for this example with triangular(0, 1, 2) arc durations are shown in Table 11.3. The CDF of the project completion time is returned as

$$F_{T_5}(t) = \begin{cases} 0 & t \leq 0 \\ \frac{1}{64512} t^{12} - \frac{1}{21504} t^{12} + \frac{5}{4032} t^{11} - \frac{1357}{120960} t^{10} + \frac{1553}{30240} t^9 - \frac{2431}{17280} t^8 + \frac{15389}{60480} t^7 - \frac{5531}{17280} t^6 + \frac{1237}{4320} t^5 - \frac{44039}{241920} t^4 + \frac{607}{7560} t^3 - \frac{13}{560} t^2 + \frac{59}{15120} t - \frac{17}{60480} & 0 < t \leq 1 \\ -\frac{5}{21504} t^{12} + \frac{5}{672} t^{11} - \frac{1813}{17280} t^{10} + \frac{8693}{10080} t^9 - \frac{556873}{120960} t^8 + \frac{1018271}{60480} t^7 - \frac{749309}{17280} t^6 + \frac{89059}{1120} t^5 - \frac{8332907}{80640} t^4 + \frac{156139}{1680} t^3 - \frac{414913}{7560} t^2 + \frac{57587}{3024} t - \frac{176249}{60480} & 1 < t \leq 2 \\ -\frac{1}{21504} t^{12} + \frac{25}{12096} t^{11} - \frac{499}{12096} t^{10} + \frac{703}{1440} t^9 - \frac{556873}{120960} t^8 + \frac{1018271}{60480} t^7 - \frac{672253}{8640} t^6 + \frac{699347}{3360} t^5 - \frac{2587009}{6720} t^4 + \frac{197879}{420} t^3 - \frac{376261}{1080} t^2 + \frac{94931}{756} t - \frac{68353}{7560} & 2 < t \leq 3 \\ \frac{1}{144} t^6 - \frac{1}{5} t^5 + \frac{19}{8} t^4 - \frac{89}{6} t^3 + \frac{409}{8} t^2 - \frac{1829}{20} t + \frac{7969}{120} & 3 < t \leq 4 \\ -\frac{1}{720} t^6 + \frac{1}{20} t^5 - \frac{3}{4} t^4 + 6 t^3 - 27 t^2 + \frac{324}{5} t - \frac{319}{5} & 4 < t \leq 5 \\ 1 & 5 < t \leq 6 \\ & t > 6. \end{cases}$$

Activity networks are useful when modeling a project having distinct activities that exhibit precedence relationships. When the durations of the activities are deterministic, it is easy to find the time needed to complete a project by using a simple longest path algorithm. However, when only random activity durations are given, the analysis is much more complicated. Monte Carlo simulation can provide an approximate solution in this case, but the simulation must be written specifically for the network and could require millions of replications for a reliable estimate. In some networks, series-parallel reduction can supply an exact solution analytically, but very few networks can be reduced in this manner. Leemis et al. [52] give an algorithm that can be applied to a more general class of networks.

11.2 Benford's Law

This section states Benford's law and shows how APPL can be used to check a probability model for conformity. The traditional analysis of Benford's law considers its applicability to *data sets*, but we consider whether *probability distributions* obey Benford's law. The material in this section is taken from Leemis et al. [53], where a more thorough treatment of the topic can be found.

Benford's law was first encountered by astronomer and mathematician Simon Newcomb, who noticed "how much faster the first pages (of tables of logarithms) wear out than the last ones" leading to the counter-intuitive conclusion that the first significant digit in the values in a logarithm table is not uniformly distributed between 1 and 9. Using a heuristic argument, he found that ones occur most often (more than 30 percent of the time) and nines least often (less than 5 percent of the time), and

$$\Pr(X = x) = \log_{10}(1 + 1/x) \quad x = 1, 2, \dots, 9,$$

where X is a random variable denoting the first significant digit. He published this result in 1881 as the "logarithm law" in the *American Journal of Mathematics*.

General Electric physicist Frank Benford [8] apparently independently arrived at the same conclusion as Newcomb concerning logarithm tables in 1938. He proceeded to "collect data from as many fields as possible" to see if natural and sociological data sets would also obey the logarithm law. He often found good agreement between the logarithm law for his 20,229 total observations, including data sets as diverse as the areas of rivers, American League baseball statistics, atomic weights of elements, death rates, and numbers appearing in *Reader's Digest*. What has become known as "Benford's law" has found applications in the distribution of the one-day return on stock market indices [55], the distribution of the populations of 3141 counties in the 1990 U.S. Census, and the detection of accounting fraud [65].

A mathematically rigorous proof of Benford's law has proven elusive. This is in part due to the fact that certain data sets (e.g., random numbers) do

not follow Benford's law. Recent attempts have considered the effect of scale invariance (e.g., dollars vs. yen), base invariance (e.g., octal vs. base ten), and mixtures (i.e., sample data drawn from several population distributions that are selected at random), as indicated in Hill [34, 35].

The purpose of this section is to switch the emphasis from the examination of *data sets* that obey Benford's law to *probability distributions* that obey Benford's law. The emphasis here is on survival distributions (i.e., distributions with positive support), although more general distributions can be examined in the same fashion. Hill [34, pages 361–362] states that “An interesting open problem is to determine which common distributions (or mixtures thereof) satisfy Benford's law . . .”. General conditions associated with the distribution of the random variable T will now be derived in order to determine when Benford's law applies.

Conditions for Conformance to Benford's Law. We refer to a random variable X with PDF

$$f_X(x) = \Pr(X = x) = \log_{10}(1 + 1/x) \quad x = 1, 2, \dots, 9$$

as a “Benford random variable.” This random variable is pre-defined in APPL as **BenfordRV**. The associated CDF is

$$F_X(x) = \Pr(X \leq x) = \log_{10}(1 + x) \quad x = 1, 2, \dots, 9.$$

Inverting the CDF, a Benford random variate X can be generated by

$$X \leftarrow \lceil 10^U - 1 \rceil,$$

or, equivalently, saving a subtraction, by

$$X \leftarrow \lfloor 10^U \rfloor,$$

where $U \sim U(0, 1)$.

Let T be a nonnegative random variable whose first significant digit is of interest. Let the integer-valued random variable D satisfy

$$10^D \leq T < 10^{D+1}$$

(e.g., $T = 365 \Rightarrow D = 2$ and $T = 1/10 \Rightarrow D = -1$). This definition of D allows the first significant digit Y of T to be written in terms of T and D as

$$Y = \lfloor T \cdot 10^{-D} \rfloor = \lfloor 10^{\log_{10} T - D} \rfloor$$

(e.g., $T = 365 \Rightarrow Y = \lfloor 365 \cdot 10^{-2} \rfloor = \lfloor 3.65 \rfloor = 3$).

Referring back to the variate generation algorithm, it is clear that if the random variable $Z = \log_{10} T - D \sim U(0, 1)$, which represents the result from the logarithm table, then the first significant digit Y has the Benford distribution. Using conditioning, the CDF of Z is given by

$$\begin{aligned}
F_Z(z) &= \Pr(Z \leq z) \\
&= \sum_{d=-\infty}^{\infty} \Pr(10^d \leq T < 10^{d+1}) \cdot \Pr(\log_{10} T - d \leq z \mid 10^d \leq T < 10^{d+1}),
\end{aligned}$$

for $0 < z < 1$. Thus conformance to Benford's law implies that the weights (the first factor in the product) associated with each order of magnitude (indexed by d) and the distribution of $Z = \log_{10} T - D$ (the second factor in the product) are such that the infinite sum produces a linear function in z .

Three examples of distributions providing a perfect fit for Benford's law are given below.

Example 11.2. Let $W \sim U(0, 2)$. Let $T = 10^W$. The PDF of T is

$$f_T(t) = \frac{1}{2t \ln 10} \quad 1 < t < 100.$$

The PDF of D is

$$f_D(0) = \Pr(D = 0) = \Pr(1 < T < 10) = \int_1^{10} f_T(t) dt = \frac{1}{2}$$

and

$$f_D(1) = \Pr(D = 1) = \Pr(10 < T < 100) = \int_{10}^{100} f_T(t) dt = \frac{1}{2}.$$

The PDF of the leading digit Y is

$$\begin{aligned}
f_Y(y) &= \Pr(Y = y) \\
&= \Pr(y < T < y + 1) + \Pr(10y < T < 10(y + 1)) \\
&= \int_y^{y+1} f_T(t) dt + \int_{10y}^{10(y+1)} f_T(t) dt \\
&= \int_y^{y+1} \frac{1}{2t \ln 10} dt + \int_{10y}^{10(y+1)} \frac{1}{2t \ln 10} dt \\
&= \log_{10} \left(\frac{y+1}{y} \right) \quad y = 1, 2, \dots, 9.
\end{aligned}$$

This PDF matches Benford's distribution exactly. Alternatively, one can proceed by determining the distribution of $Z = \log_{10} T - D$ as follows, where $W = \log_{10} T$:

$$\begin{aligned}
F_Z(z) &= \Pr(Z \leq z) \\
&= \sum_{d=-\infty}^{\infty} \Pr(10^d \leq T < 10^{d+1}) \cdot \Pr(\log_{10} T - d \leq z \mid 10^d \leq T < 10^{d+1})
\end{aligned}$$

$$\begin{aligned}
&= \sum_{d=0}^1 \Pr(d \leq W < d+1) \cdot \Pr(W - d \leq z | d \leq W < d+1) \\
&= \Pr(0 \leq W < 1) \cdot \Pr(W \leq z | 0 \leq W < 1) + \\
&\quad \Pr(1 \leq W < 2) \cdot \Pr(W - 1 \leq z | 1 \leq W < 2) \\
&= \frac{1}{2}z + \frac{1}{2}z \\
&= z \quad 0 < z < 1.
\end{aligned}$$

Since this is the CDF for a $U(0, 1)$ random variable, Benford's law is satisfied exactly.

This example can be generalized as follows. Let $W \sim U(a, b)$, where a and b are real numbers satisfying $a < b$. If the interval $(10^a, 10^b)$ covers an integer number of orders of magnitude, then the first significant digit of the random variable $T = 10^W$ satisfies Benford's law exactly. Equivalently, if $b - a$ is a positive integer, then the first significant digit of $T = 10^W$ satisfies Benford's law. Examples include $a = -2$, $b = 1$ and $a = \log_{10}(3/2)$, $b = \log_{10}(150)$, with $(10^a, 10^b)$ equal to $(0.01, 10)$ and $(1.5, 150)$ respectively.

There is no need for the support of the distribution of $T = 10^W$ to span several orders of magnitude as is the case for many of the data sets that conform to Benford's law. A single order of magnitude (e.g., $a = 5$, $b = 6$) is sufficient.

The next example considers a non-uniform distribution for W that satisfies Benford's law exactly.

Example 11.3. Let $W \sim \text{triangular}(0, 1, 2)$. The PDF for W is

$$f_W(w) = \begin{cases} w & 0 < w < 1 \\ 2 - w & 1 \leq w < 2. \end{cases}$$

As before, let $T = 10^W$ and $Z = W - D$. The CDF of Z is

$$\begin{aligned}
F_Z(z) &= \sum_{d=0}^1 \Pr(d \leq W < d+1) \cdot \Pr(W - d \leq z | d \leq W < d+1) \\
&= \frac{1}{2}z^2 + \frac{1}{2}(2z - z^2) \\
&= z \quad 0 < z < 1.
\end{aligned}$$

Thus the first significant digit of T satisfies Benford's law exactly.

This example can also be generalized. Let $W \sim \text{triangular}(a, b, c)$, where a , b , and c are real numbers satisfying $a < b < c$. The first significant digit of the random variable $T = 10^W$ satisfies Benford's law exactly if a , b , and c are integers. The next example considers a non-symmetric distribution for W .

Example 11.4. Let W have PDF

$$f_W(w) = \begin{cases} 1 - w^2 & -1 < w < 0 \\ (w - 1)^2 & 0 \leq w < 1. \end{cases}$$

As before, let $T = 10^W$ and $Z = W - D$. The CDF of Z is

$$\begin{aligned} F_Z(z) &= \sum_{d=-1}^0 \Pr(d \leq W < d+1) \cdot \Pr(W - d \leq z \mid d \leq W < d+1) \\ &= \frac{2}{3} \left[-\frac{z^3}{2} + \frac{3z^2}{2} \right] + \frac{1}{3} [1 + (z-1)^3] \\ &= z \quad 0 < z < 1. \end{aligned}$$

Thus the first significant digit of T satisfies Benford's law exactly.

This example can be generalized for W with PDF

$$f_W(w) = \begin{cases} 1 - w^n & -1 < w < 0 \\ (w - 1)^n & 0 \leq w < 1, \end{cases}$$

where n is a positive, even integer.

In order to test other probability distributions for conformance to Benford's law, we automated the process of deriving the distribution of Z by writing the APPL procedure **Benford**, whose argument is a random variable representing W and whose returned value is the distribution of Z . The algorithm is shown below.

Procedure Benford: Compute the distribution of Z to check conformity to Benford's law.

Input: A random variable W formatted in the usual APPL list-of-sublists format.

Output: The PDF of Z .

Algorithm:

```

 $\Omega \leftarrow \text{Support}(W)$            the set  $\Omega$  is the support of the random variable  $W$ 
 $Lo \leftarrow \lfloor \Omega \rfloor$            lower loop limit
 $Hi \leftarrow \lceil \Omega \rceil - 1$     upper loop limit
Dimension  $W[Hi - Lo + 1]$         $W$  holds the mixture probabilities
Dimension  $TranW[Hi - Lo + 1]$     $TranW$  holds transformed segments of  $W$ 
For  $d \leftarrow Lo$  to  $Hi$ 
     $W[d] \leftarrow F_W(d+1) - F_W(d)$     calculate weights for the mixture
     $TruncW[d] \leftarrow \text{Truncate}(W, d, d+1)$   truncate  $W$  between  $d$  and  $d+1$ 
     $TranW[d] \leftarrow \text{Transform}(TruncW, w - d)$   shift  $W$  by  $d$  units
 $Z \leftarrow \text{Mixture}(W, TranW)$     compute the PDF of the mixture

```

The APPL code required to return the distribution of Z for Example 11.2 is

```
> W := UniformRV(0, 2);
> Z := Benford(W);
```

The APPL code required to return the distribution of Z for Example 11.3 is

```
> W := TriangularRV(0, 1, 2);
> Z := Benford(W);
```

The APPL code required to return the distribution of Z for Example 11.4 is

```
> W := [[w -> 1 - w ^ 2, w -> (1 - w) ^ 2],
        [-1, 0, 1], ["Continuous", "PDF"]];
> Z := Benford(W);
```

All three code segments return $Z \sim U(0, 1)$.

11.3 Miscellaneous Applications

This final section contains a series of short examples that are appropriate for the symbolic capabilities of APPL that were not adequately covered in previous chapters.

Example 11.5. The central limit theorem can be used to facilitate inference about the population mean. The central limit theorem implies

that $\bar{X} = \frac{1}{n} \sum_{i=1}^n X_i$ is asymptotically normally distributed when the

X_i 's are iid random variables with finite mean and variance. Application of the central limit theorem becomes restricted, however, with small sample sizes, with skewed data, or in the presence of outliers. In these cases, an alternative method is needed to determine the exact distribution of \bar{X} . Once the distribution \bar{X} is found, one can use it to make statistical inferences about the population mean. Consider the case of the population associated with $X_i \sim \text{exponential}(1)$. By the central limit theorem, the distribution of $Y = \bar{X}$ is approximately $N(1, \frac{1}{\sqrt{n}})$ for large n . For the case $n = 3$, one can compute the exact distribution of Y with the following statements:

```
> X := ExponentialRV(1);
> n := 3;
> Y := ConvolutionIID(X, n);
> Y := Transform(Y, [[x -> x / n], [0, infinity]]);
```

The resulting Erlang PDF is

$$f_Y(y) = \frac{27}{2} y^2 e^{-3y} \quad y > 0,$$

Table 11.4. Fractiles for exact and approximated distributions.

Approach	Distribution	Fractile			
		0.9	0.95	0.975	0.99
Exact	\bar{X}	1.240	1.318	1.388	1.473
Central limit theorem	$N(1, \frac{1}{\sqrt{30}})$	1.234	1.300	1.358	1.425

which is predictably non-normal in shape since n is too small for the normality of the central limit theorem to apply to such a skewed underlying distribution. Now consider the case of $n = 30$, often considered a sufficiently large sample size for an adequate central limit theorem approximation. Here a computation of the error associated with the approximation is provided. To obtain the PDF of \bar{X} , simply change the second line of code to `n := 30`; The result is the following Erlang PDF:

$$f_Y(y) = \frac{246315313339233398437500000}{10577732774609} y^{29} e^{-30y} \quad y > 0.$$

Table 11.4 contains selected fractiles for both the exact distribution of \bar{X} and its central limit theorem approximation, the $N(1, \frac{1}{\sqrt{30}})$ distribution. For the selected fractiles, the approximation is only good to about one digit beyond the decimal point. The additional APPL statements that generate the central limit theorem values in Table 11.4 are

```
> Z := NormalRV(1, 1 / sqrt(n));
> alpha := [0.9, 0.95, 0.975, 0.99];
> for i from 1 to 4 do
>   evalf(IDF(Z, alpha[i]));
> od;
```

Example 11.6. As a second application related to the central limit theorem, consider an old, approximate method of generating random variates that have a standard normal distribution:

$$Z^* = U_1 + U_2 + \cdots + U_{12} - 6,$$

where the U_i 's are iid $U(0, 1)$ random variables. The following APPL statements determine the exact distribution of this 12-fold sum:

```
> U := UniformRV(0, 1);
> U12 := ConvolutionIID(U, 12);
> Zstar := Transform(U12, [[x -> x - 6], [0, 12]]);
```

which returns the PDF of Z^* as

$$f_{Z^*}(x) = \left\{ \begin{array}{ll} \frac{1}{39916800} (x+6)^{11} & -6 < x < -5 \\ -\frac{1}{3628800} x^{11} - \frac{1}{67200} x^{10} - \frac{11}{30240} x^9 - \frac{107}{20160} x^8 - \frac{517}{10080} x^7 - \\ \frac{2477}{7200} x^6 - \frac{11737}{7200} x^5 - \frac{54797}{10080} x^4 - \frac{250657}{20160} x^3 - \frac{1113317}{60480} x^2 - \\ \frac{4726777}{302400} x - \frac{18595037}{3326400} & -5 < x < -4 \\ \frac{1}{725760} x^{11} + \frac{1}{17280} x^{10} + \frac{11}{10080} x^9 + \frac{7}{576} x^8 + \frac{99}{1120} x^7 + \\ \frac{631}{1440} x^6 + \frac{1199}{800} x^5 + \frac{1009}{288} x^4 + \frac{12199}{2240} x^3 + \frac{9385}{1728} x^2 + \\ \frac{38533}{11200} x + \frac{894727}{665280} & -4 < x < -3 \\ -\frac{1}{241920} x^{11} - \frac{1}{8064} x^{10} - \frac{11}{6720} x^9 - \frac{25}{2016} x^8 - \frac{33}{560} x^7 - \\ \frac{13}{72} x^6 - \frac{143}{400} x^5 - \frac{239}{504} x^4 - \frac{583}{1120} x^3 - \frac{1619}{3024} x^2 - \\ \frac{781}{5600} x + \frac{61297}{166320} & -3 < x < -2 \\ \frac{1}{120960} x^{11} + \frac{1}{6720} x^{10} + \frac{11}{10080} x^9 + \frac{1}{252} x^8 + \frac{11}{1680} x^7 + \\ \frac{1}{360} x^6 + \frac{11}{1200} x^5 + \frac{25}{504} x^4 + \frac{11}{3360} x^3 - \frac{563}{3024} x^2 + \\ \frac{11}{50400} x + \frac{65521}{166320} & -2 < x < -1 \\ -\frac{1}{86400} x^{11} - \frac{1}{14400} x^{10} + \frac{1}{1440} x^8 - \frac{23}{3600} x^6 + \frac{31}{720} x^4 - \\ \frac{809}{4320} x^2 + \frac{655177}{1663200} & -1 < x < 0 \\ \frac{1}{86400} x^{11} - \frac{1}{14400} x^{10} + \frac{1}{1440} x^8 - \frac{23}{3600} x^6 + \frac{31}{720} x^4 - \\ \frac{809}{4320} x^2 + \frac{655177}{1663200} & 0 < x < 1 \\ -\frac{1}{120960} x^{11} + \frac{1}{6720} x^{10} - \frac{11}{10080} x^9 + \frac{1}{252} x^8 - \frac{11}{1680} x^7 + \\ \frac{1}{360} x^6 - \frac{11}{1200} x^5 + \frac{25}{504} x^4 - \\ \frac{11}{3360} x^3 - \frac{563}{3024} x^2 - \frac{11}{50400} x + \frac{65521}{166320} & 1 < x < 2 \\ \frac{1}{241920} x^{11} - \frac{1}{8064} x^{10} + \frac{11}{6720} x^9 - \frac{25}{2016} x^8 + \frac{33}{560} x^7 - \\ \frac{13}{72} x^6 + \frac{143}{400} x^5 - \frac{239}{504} x^4 + \frac{583}{1120} x^3 - \frac{1619}{3024} x^2 + \\ \frac{781}{5600} x + \frac{61297}{166320} & 2 < x < 3 \\ -\frac{1}{725760} x^{11} + \frac{1}{17280} x^{10} - \frac{11}{10080} x^9 + \frac{7}{576} x^8 - \frac{99}{1120} x^7 + \\ \frac{631}{1440} x^6 - \frac{1199}{800} x^5 + \frac{1009}{288} x^4 - \frac{12199}{2240} x^3 + \frac{9385}{1728} x^2 - \\ \frac{38533}{11200} x + \frac{894727}{665280} & 3 < x < 4 \\ \frac{1}{3628800} x^{11} - \frac{1}{67200} x^{10} + \frac{11}{30240} x^9 - \frac{107}{20160} x^8 + \frac{517}{10080} x^7 - \\ \frac{2477}{7200} x^6 + \frac{11737}{7200} x^5 - \frac{54797}{10080} x^4 + \frac{250657}{20160} x^3 - \frac{1113317}{60480} x^2 + \\ \frac{4726777}{302400} x - \frac{18595037}{3326400} & 4 < x < 5 \\ -\frac{1}{39916800} x^{11} + \frac{1}{604800} x^{10} - \frac{1}{20160} x^9 + \frac{1}{1120} x^8 - \frac{3}{280} x^7 + \\ \frac{9}{100} x^6 - \frac{27}{50} x^5 + \frac{81}{35} x^4 - \frac{243}{35} x^3 + \frac{486}{35} x^2 - \\ \frac{2916}{175} x + \frac{17496}{1925} & 5 < x < 6. \end{array} \right.$$

The following additional APPL statements calculate the kurtosis of Z^* and the kurtosis of a standard normal random variable Z :

```
> Z := NormalRV(0, 1);
> Kurtosis(Zstar);
> Kurtosis(Z);
```

The distribution of Z^* is useful because, like the standard normal distribution, its mean is zero, its variance is one, and its skewness is zero. However, the fourth moments of Z^* and Z differ, with

$$E \left[\left(\frac{Z^* - E[Z^*]}{\sqrt{V[Z^*]}} \right)^4 \right] = E[(Z^*)^4] = 2.9,$$

while the kurtosis of Z is 3. An overlaid Maple plot of the standard normal PDF and $f_{Z^*}(x)$ reveals that this approximation does quite well. It is apparent that the PDFs match very nicely except around $x = 0$ [where $f_Z(0) = \frac{1}{\sqrt{2\pi}} \cong 0.399$ and $f_{Z^*}(0) = \frac{655177}{1663200} \cong 0.394$] and, of course, in the tails, since the approximate distribution only has support on $(-6, 6)$. To obtain overlaid plots of these two distributions, use the `PlotDist` procedure. First create a plot of the PDF of Z^* and store it as `Plot1`. Next, create a plot of the PDF of Z and store it as `Plot2`. Finally, use the `plots[display]` procedure to combine the two plots. This is summarized in the statements

```
> Plot1 := PlotDist(Zstar, -6, 6):
> Plot2 := PlotDist(Z, -6, 6):
> plots[display]({Plot1, Plot2}, scaling = unconstrained);
```

Dozens of tables of mathematically intractable statistical values have been published over the years. Some of these constants can be calculated in APPL, as illustrated in the next example.

Example 11.7. Balakrishnan and Chen's [5] text is a valuable resource for comparing various measures associated with probability distributions. Here we use APPL's capabilities to produce similar measures. This book begins with 39 pages of theory and references to the inverse Gaussian (IG) distribution and its order statistics, followed by 645 pages of PDF plots, expected value tables, covariance tables, estimation tables, and so forth. APPL is able to create a number of these figures and tables, and in some cases goes beyond those presented in the text. Page 50, for example, shows the $N(0, 1)$ and the standardized IG(0.8) PDFs overlaid in a single plot. The PDF of a standardized IG random variable X [which is listed as equation (2.14) on page 7 of the text] is

$$f_X(x) = \frac{1}{\sqrt{2\pi}} \left(\frac{3}{3+kx} \right)^{3/2} e^{-3x^2/(6+2kx)} - \frac{3}{k} < x < \infty.$$

The two plots may be overlaid and animated for k increasing from zero to one as follows:

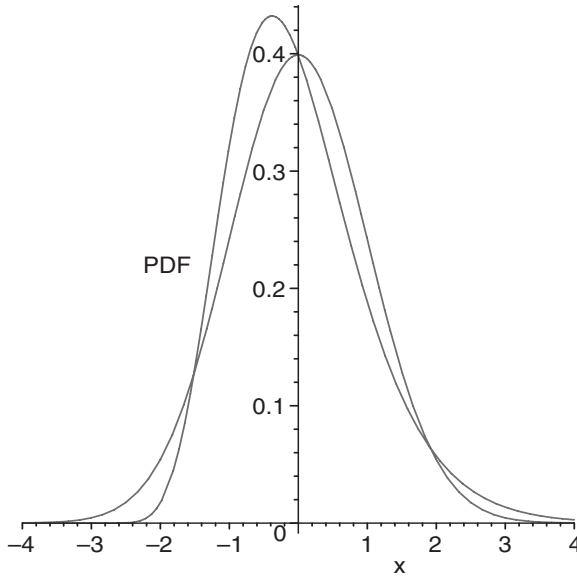


Fig. 11.6. Overlaid plots of the standard normal and standard IG(0.8) distributions.

```

> Z := NormalRV(0, 1);
> X := [[x -> (3 / (3 + k * x)) ^ (3 / 2) *
          exp(-3 * x ^ 2 / (6 + 2 * k * x)) /
          sqrt(2 * Pi)],
         [-3 / k, infinity], ["Continuous", "PDF"]];
> NormalExpr := op(unapply(Z[1](x))(x));
> InvGaussExpr := op(unapply(X[1](x))(x));
> unassign('k');
> plots[animate]({NormalExpr, InvGaussExpr},
                  x = -4 .. 4, k = 0 .. 1);

```

The plot is shown in Figure 11.6 for $k = 0.8$. To execute the animation, first select the plot by clicking on it. Then choose “Play” from the “Animation” menu. Balakrishnan and Chen use such plots to show how the two distributions separate as k increases from zero. Clearly APPL is not limited to plotting the standardized version of the IG PDF, but can be used to plot IG PDF, CDF, HF, and so on. Their text also gives a listing of selected means of order statistics for the standardized IG distribution Z . Using APPL one can calculate, for example, the mean of the order statistic $Z_{(2)}$ for $n = 4$ (when $k = 0.7$, as on page 74 of their text) as follows:

```

> k := 0.7;
> Z := [[z -> 1 / sqrt(2 * Pi) * (3 / (3 + k * z)) ^

```

```

      (3 / 2) * exp(-3 * z ^ 2 /
      (2 * (3 + k * z))))],
      [-3 / k, infinity], ["Continuous", "PDF"]];
> Zos := OrderStat(Z, 4, 2);
> avg := Mean(Zos);

```

which returns the value -0.35202 in agreement with the authors. In addition to giving the mean, note that the variable `Zos` above is in fact the list-of-sublists representation of the distribution of that order statistic. Thus one might use any number of functions on that variable, in order to find critical points, significance levels, and can use that distribution in determining other distributions, e.g., by invoking `Product` or `Convolution`.

Another resource, such as Harter and Balakrishnan's [32] handbook of tables for the use of order statistics in estimation also gives extensive charts for various applications of order statistics. APPL can be used in a similar fashion to duplicate many of their tabled values, and, in some cases, go beyond the tabled values. See Glen et al. [29] for a specific example.

Example 11.8. Define a geometric random variable X with parameter $p = \frac{1}{4}$ to model the number of trials up to and including the first success, i.e.,

$$f_X(x) = \frac{1}{4} \cdot \left(\frac{3}{4}\right)^{x-1} \quad x = 1, 2, \dots$$

Calculate the median of the maximum order statistic when $n = 5$ items are sampled *with replacement* from this geometric distribution. The APPL statements

```

> X := GeometricRV(1 / 4);
> Y := OrderStat(X, 5, 5);
> IDF(Y, 0.5);

```

return the median of the distribution as 8. The final statement could have equivalently been issued as `Median(Y)`;

Example 11.9. A sequence of order statistics drawn from a unit exponential distribution was considered in Example 3.11. Five plots were generated corresponding to the PDF of each of the order statistics via the APPL code

```

> X := ExponentialRV(1);
> n := 5;
> for i from 1 to n do
>   PlotDist(OrderStat(X, n, i), 0, 6);
> od;

```

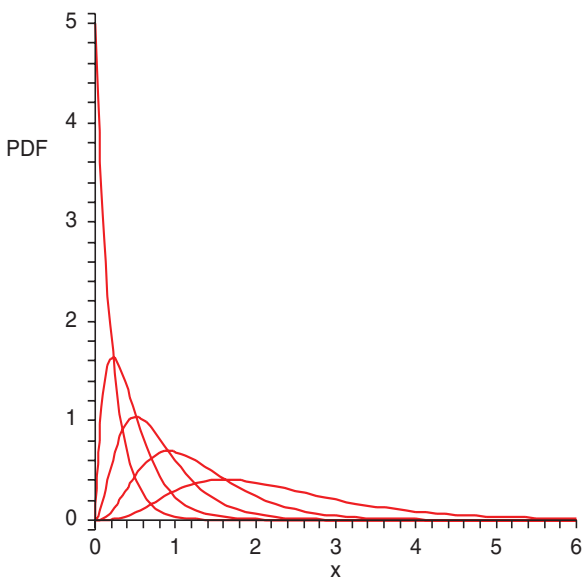


Fig. 11.7. Overlaid plots of the PDFs of order statistics drawn from an exponential population.

But what if one wanted to see all five of the order statistics on a single graph? This can be done by modifying the code to

```
> X := ExponentialRV(1);
> n := 5;
> OSgraphs := [];
> for i from 1 to n do
>   OSgraphs := [op(OSgraphs),
                  PlotDist(OrderStat(X, n, i), 0, 6)]:
> od:
> with(plots);
> display(OSgraphs);
```

This code initializes `OSgraphs` to an empty list, and concatenates the various graphs in the loop. The `with(plots)` command is used to include additional Maple plotting options. Finally, `display` is used to plot multiple graphs on one plot. The graph is shown in Figure 11.7.

References

1. Adlakha, V.G., Kulkarni, V.G. (1989), "A Classified Bibliography of Research on Stochastic PERT Networks: 1966–1987," *INFOR*, Volume 27, 272–296.
2. Andrews, D.W.K., Buchinsky, M. (2000), "A Three-Step Method for Choosing the Number of Bootstrap Repetitions," *Econometrica*, Volume 68, 23–51.
3. Andrews, D.W.K., Buchinsky, M. (2002), "On the Number of Bootstrap Repetitions for BC_a Confidence Intervals," *Econometric Theory*, Volume 18, 962–984.
4. Arnold, B.C., Balakrishnan, N., Nagaraja, H.N. (1992), *A First Course in Order Statistics*, John Wiley & Sons.
5. Balakrishnan, N., Chen, W.W.S. (1997), *CRC Handbook of Tables for Order Statistics from Inverse Gaussian Distributions with Applications*, CRC Press.
6. Banks, J., Carson, J.S., Nelson, B.L., Nicol, D.M. (2005), *Discrete-Event System Simulation*, 4th ed., Prentice-Hall.
7. Barr, D., Zehna, P.W. (1971), *Probability*, Brooks/Cole.
8. Benford, F. (1938), "The Law of Anomalous Numbers," *Proceedings of the American Philosophical Society*, Volume 78, 551–572.
9. Birnbaum, Z.W. (1952), "Numerical Tabulation of the Distribution of Kolmogorov's Statistic for Finite Sample Size," *Journal of the American Statistical Association*, Volume 47, 425–441.
10. Burr, I.W. (1955), "Calculation of Exact Sampling Distribution of Ranges from a Discrete Population," *Annals of Mathematical Statistics*, Volume 26, 530–532. Correction, Volume 38, 280.
11. Carrano, F.M., Helman, P., Veroff, R. (1998), *Data Abstraction and Problem Solving with C++: Walls and Mirrors*, 2nd ed., Addison-Wesley Longman.
12. Casella, G., Berger, R. (2002), *Statistical Inference*, 2nd ed., Duxbury.
13. Ciardo, G., Leemis, L., Nicol, D. (1995), "On the Minimum of Independent Geometrically Distributed Random Variables," *Statistics and Probability Letters*, Volume 23, 313–326.
14. D'Agostino, R.B., Stephens, M.A. (1986), *Goodness-of-Fit Techniques*, Marcel Dekker.
15. David, H.A., Nagaraja, H.N. (2003), *Order Statistics*, 3rd ed., John Wiley & Sons.
16. Devroye, L. (1996), "Random Variate Generation in One Line of Code," *Proceedings of the 1996 Winter Simulation Conference*, Charnes, J., Morrice, D., Brunner, D., Swain, J., eds., Institute of Electrical and Electronics Engineers, 265–272.

17. Drew, J., Glen, A., Leemis, L. (2000), "Computing the Cumulative Distribution Function of the Kolmogorov–Smirnov Statistic," *Computational Statistics and Data Analysis*, Volume 34, 1–15.
18. Duggan, M., Drew, J., Leemis, L. (2005), "A Test of Randomness Based on the Distance Between Consecutive Random Number Pairs," *Proceedings of the 2005 Winter Simulation Conference*, Kuhl, M.E., Steiger, N.M., Armstrong, F.B., Joines, J.A., eds., Institute of Electrical and Electronics Engineers, 741–748.
19. Efron, B., Tibshirani, R.J. (1993), *An Introduction to the Bootstrap*, Chapman & Hall.
20. Elmaghraby, S.E. (1977), *Activity Networks: Project Planning and Control by Network Models*, John Wiley & Sons.
21. Evans, D., Drew, J., Leemis, L. (2007), "The Distribution of the Kolmogorov–Smirnov, Cramer–von Mises, and Anderson–Darling Test Statistics for Exponential Populations with Estimated Parameters," Technical Report, Department of Mathematics, The College of William & Mary.
22. Evans, D., Leemis, L. (2000), "Input Modeling Using a Computer Algebra System," *Proceedings of the 2000 Winter Simulation Conference*, Joines, J., Barton, R., Fishwick, P., Kang, K., eds., Institute of Electrical and Electronics Engineers, 577–586.
23. Evans, D., Leemis, L. (2004), "Algorithms for Determining the Distributions of Sums of Discrete Random Variables," *Mathematical and Computer Modelling*, Volume 40, 1429–1452.
24. Evans, D., Leemis, L., Drew, J. (2006), "The Distribution of Order Statistics for Discrete Random Variables with Applications to Bootstrapping," *INFORMS Journal on Computing*, Volume 18, 19–30.
25. Fisher, D.L., Saisi, D., Goldstein, W.M. (1985), "Stochastic PERT Networks: OP Diagrams, Critical Paths and the Project Completion Time," *Computers and Operations Research*, Volume 12, 471–482.
26. Fishman, G.S. (2001), *Discrete-Event Simulation: Modeling, Programming, and Analysis*, Springer.
27. Gehan, E.A. (1965), "A Generalized Wilcoxon Test for Comparing Arbitrarily Singly-Censored Samples," *Biometrika*, Volume 52, Parts 1 and 2, 203–223.
28. Glen, A., Drew, J., Leemis, L. (1997), "A Generalized Univariate Change-of-Variable Transformation Technique," *INFORMS Journal on Computing*, Volume 9, 288–295.
29. Glen, A., Evans, D., Leemis, L. (2001), "APPL: A Probability Programming Language," *The American Statistician*, Volume 55, 156–166.
30. Glen, A., Leemis, L., Drew, J. (2004), "Computing the Distribution of the Product of Two Continuous Random Variables," *Computational Statistics and Data Analysis*, Volume 44, 451–464.
31. Grinstead, C.M., Snell, J.L. (1997), *Introduction to Probability*, 2nd rev. ed., American Mathematical Society.
32. Harter, H.L., Balakrishnan, N. (1996), *CRC Handbook of Tables for the Use of Order Statistics in Estimation*, CRC Press.
33. Hasting, K.J. (2006), *Introduction to the Mathematics of Operations Research with Mathematica*, 2nd ed., CRC Press.
34. Hill, T.P. (1995), "A Statistical Derivation of the Significant-Digit Law," *Statistical Science*, Volume 86, 354–363.
35. Hill, T.P. (1998), "The First Digit Phenomenon," *American Scientist*, Volume 86, 358–363.

36. Hogg, R.V., Craig, A.T. (1995), *Introduction to the Mathematical Statistics*, 5th ed., Prentice-Hall.
37. Hogg, R.V., McKean, J.W., Craig, A.T. (2005), *Introduction to the Mathematical Statistics*, 6th ed., Prentice-Hall.
38. Hogg, R.V., Tanis, E.A. (2001), *Probability and Statistical Inference*, 6th ed., Prentice-Hall.
39. Hutson, A.D., Ernst, M.D. (2000), "The Exact Bootstrap Mean and Variance of an L -Estimator," *Journal of the Royal Statistical Society, Series B*, Volume 62, 89–94.
40. Johnson, N.L., Kotz, S., Balakrishnan, N. (1995), *Continuous Univariate Distributions*, Volume 2, 2nd ed., John Wiley & Sons.
41. Kalbfleisch, J.D., Prentice, R.L. (2002), *The Statistical Analysis of Failure Time Data*, 2nd ed., John Wiley & Sons.
42. Karian, Z.A., Tanis, E.A. (1999), *Probability and Statistics: Explorations with Maple*, 2nd ed., Prentice-Hall.
43. Knuth, D.E. (1998), *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, 3rd ed., Addison-Wesley.
44. L'Écuyer, P., Cordeau, J.-F., Simard, R. (2000), "Close-Point Spatial Tests and Their Application to Random Number Generators," *Operations Research*, Volume 48, 308–317.
45. Laplante, P.A., ed. (2001), *Dictionary of Computer Science, Engineering and Technology*, CRC Press.
46. Larsen, R.J., Marx, M.L. (2001), *An Introduction to Mathematical Statistics and Its Applications*, 3rd ed., Prentice-Hall.
47. Larsen, R.J., Marx, M.L. (2006), *An Introduction to Mathematical Statistics and Its Applications*, 4th ed., Prentice-Hall.
48. Law, A.M. (2007), *Simulation Modeling and Analysis*, 4th ed., McGraw-Hill.
49. Lawless, J.F. (2003), *Statistical Models and Methods for Lifetime Data*, 2nd ed., John Wiley & Sons.
50. Leemis, L. (1995), *Reliability: Probabilistic Models and Statistical Methods*, Prentice-Hall.
51. Leemis, L. (2006), "Lower Confidence Bounds for System Reliability from Binary Failure Data Using Bootstrapping," *Journal of Quality Technology*, Volume 38, 2–13.
52. Leemis, L., Duggan, M., Drew, J., Mallozzi, J., Connell, K. (2006), "Algorithms to Calculate the Distribution of the Longest Path Length of a Stochastic Activity Network with Continuous Activity Durations," *Networks*, Volume 48, 143–165.
53. Leemis, L., Schmeiser, B., Evans, D. (2000), "Survival Distributions Satisfying Benford's Law," *The American Statistician*, Volume 54, 236–241.
54. Lehmer, D.H. (1951), "Mathematical Methods in Large-Scale Computing Units," *Proceedings of the 2nd Symposium on Large-Scale Calculating Machinery*, Harvard University Press, 141–146.
55. Ley, E. (1996), "On the Peculiar Distribution of the U.S. Stock Indices Digits," *The American Statistician*, Volume 50, 311–313.
56. Lieblein, J., Zelen, M. (1956), "Statistical Investigation of the Fatigue Life of Deep-Groove Ball Bearings," *Journal of Research of the National Bureau of Standards*, Volume 57, 273–316.

57. Margolin, B.H., Winokur, H.S. (1967), "Exact Moments of the Order Statistics of the Geometric Distribution and their Relation to Inverse Sampling and Reliability of Redundant Systems," *Journal of the American Statistical Association*, Volume 62, 915–925.
58. Marsaglia, G. (1968), "Random Numbers Fall Mainly in the Planes," *Proceedings of the National Academy of Sciences*, Volume 61, 25–28.
59. Martin, J.J. (1965), "Distribution of the Time Through a Directed, Acyclic Network," *Operations Research*, Volume 13, 44–66.
60. Martin, M.A. (1990), "On Bootstrap Iteration for Coverage Correction in Confidence Intervals," *Journal of the American Statistical Association* Volume 85, 1105–1118.
61. Miller, L.H. (1956), "Table of Percentage Points of Kolmogorov Statistics," *Journal of the American Statistical Association*, Volume 51, 111–121.
62. Miller, I., Miller, M. (2004), *John E. Freund's Mathematical Statistics*, 7th ed., Prentice–Hall.
63. Nelson, B.L., Yamnitsky, M. (1998), "Input Modeling Tools for Complex Problems," *Proceedings of the 1998 Winter Simulation Conference*, Medeiros, D.J., Watson, E.F., Carson, J.S., Manivannan, M.S., eds., Institute of Electrical and Electronics Engineers, 105–112.
64. Nicol, D. (2000), personal communication.
65. Nigrini, M. (1996), "A Taxpayer Compliance Application of Benford's Law," *Journal of the American Taxation Association*, Volume 18, 72–91.
66. Owen, D.B. (1962), *Handbook of Statistical Tables*, Addison–Wesley.
67. Padgett, W.J., Tomlinson, M.A. (2003), "Lower Confidence Bounds for Percentiles of Weibull and Birnbaum–Saunders Distributions," *Journal of Statistical Computation and Simulation*, Volume 73, 429–443.
68. Park, S.K., Miller, K.W. (1988), "Random Number Generators: Good Ones Are Hard to Find," *Communications of the ACM*, Volume 31, 1192–1201.
69. Parlar, M. (2000), *Interactive Operations Research with Maple*, Birkhäuser.
70. Parzen, E. (1960), *Modern Probability Theory and Its Applications*, John Wiley & Sons.
71. Port, S.C. (1994), *Theoretical Probability for Applications*, John Wiley & Sons.
72. Rice, J.A. (1995), *Mathematical Statistics and Data Analysis*, 2nd ed., Wadsworth.
73. Rohatgi, V.K. (1976), *An Introduction to Probability Theory Mathematical Statistics*, John Wiley & Sons.
74. Rose, C., Smith, M.D. (2002), *Mathematical Statistics and Mathematica*, Springer–Verlag.
75. Ross, S. (2006), *A First Course In Probability*, 7th ed., Prentice–Hall.
76. Shier, D.R. (1991), *Network Reliability and Algebraic Structures*, Oxford University Press.
77. Springer, M.D. (1979), *The Algebra of Random Variables*, John Wiley & Sons.
78. Srivastava, R.C. (1974), "Two Characterizations of the Geometric Distribution," *Journal of the American Statistical Association*, Volume 69, 267–269.
79. Thompson, P. (2000), "Getting Normal Probability Approximations Without Using Normal Tables," *The College of Mathematics Journal*, Volume 31, 51–54.
80. Trosset, M. (2001), personal communication.
81. Weiss, M.A. (1994), *Data Structures and Algorithm Analysis in C++*, Addison–Wesley Publishing Company.

82. Woodward, J.A., Palmer, C.G.S. (1997), "On the Exact Convolution of Discrete Random Variables," *Applied Mathematics and Computation*, Volume 83, 69–77.
83. Young, D.H. (1970), "The Order Statistics of the Negative Binomial Distribution," *Biometrika*, Volume 57, 181–186.

Index

- Active cell, 98, 100–102
- Activity network, 185
- Actuarial science and survival analysis, 144
- Algorithm for procedure **Convolution**, 106–108
- Algorithm for procedure **Product**, 65–68
- APPL. *See* A Probability Programming Language
- APPL (Maple) commands
 - assume**, 90
 - BenfordRV**, 195
 - BinomialRV**, 89, 143
 - CDF**, 4, 37–39, 44, 88, 154, 158, 161, 179, 190, 191
 - CHF**, 39, 44
 - combine**, 16
 - Convolution**, 65, 91–95, 106–117, 140, 158, 190
 - ConvolutionIID**, 4, 44, 92, 113, 116, 148, 199–200
 - diff**, 24
 - Difference**, 158, 188, 191
 - evalf**, 14, 25
 - evalhf**, 14
 - expand**, 17
 - ExpectedValue**, 41, 44, 89
 - ExponentialRV**, 39, 41–43, 137, 151, 161, 162, 164, 199, 204, 205
 - fsolve**, 20–21, 163
 - GammaRV**, 140, 161, 167
 - HF**, 39, 44, 136
 - IDF**, 38, 39, 44, 138, 139, 150, 151, 200, 204
 - KSRV**, 156, 176, 179, 181
 - Kurtosis**, 41, 44, 62, 89, 201
 - limit**, 14
 - LognormalRV**, 62
 - Maximum**, 44, 138, 191
 - Mean**, 41, 44, 62, 113, 129, 136, 137, 204
 - member**, 19, 20
 - MGF**, 41, 44, 107
 - Minimum**, 44, 138
 - Mixture**, 44, 198
 - MLE**, 44, 151, 161–164, 166
 - MOM**, 44, 167
 - nargs**, 28
 - NegativeBinomialRV**, 89
 - nops**, 19
 - NormalRV**, 38, 41, 63, 200–203
 - OrderStat**, 5, 6, 42–44, 119–131, 136, 146, 204, 206
 - PDF**, 5, 6, 39, 44, 83, 86–87, 90, 113, 116, 122, 125, 179
 - PlotDist**, 4, 40, 43, 88, 179, 202, 204, 205
 - PoissonRV**, 110
 - proc**, 28
 - Product**, 4, 55–68, 143
 - ProductIID**, 7, 65
 - restart**, 15
 - seq**, 20
 - SF**, 7, 38, 39, 44, 135–136, 140, 156
 - simplify**, 17

- Skewness, 41, 44, 62, 89, 160–161
- sum, 25
- Transform, 9, 44, 48–54, 64, 143, 148, 158, 162, 198–200
- TriangularRV, 4, 40, 41, 199
- Truncate, 44, 138, 198
- unapply, 17, 24
- UniformDiscreteRV, 5, 6, 113, 122, 128
- UniformRV, 4, 9, 37, 50, 52, 61, 64, 65, 138, 139, 154, 158, 199, 200
- Variance, 41, 44, 62, 89, 113, 129, 136, 146, 148
- VerifyPDF, 9, 35, 44, 90
- WeibullRV, 38, 136, 163
- with, 18
- with(linalg), 18
- with(networks), 18
- APPL procedure code
 - calculating bootstrap standard deviation of median, 146
 - calculating PDF of D_n , 158
 - computation of distribution of median, 124
 - distribution of failure time, computation of, 138
 - exact value of standard error of mean, calculation of, 148
 - moments estimates, 167
 - PDF, computation of, 122
 - probability ‘that the mission is successful’ calculation, 140
 - required to return distribution of Z , 199
 - statements, 129
- A Probability Programming Language
 - list-of-sublists data structure, 106–107, 111
 - uses of, 3–8
- Arc’s criticality, 186
- Arcsin distribution, 54
- Arrays, in Maple, 20
- Arrays s and $Probs$ for sums of discrete random variables, 96
- BallBearing, 161
- Benford random variable, 80
- Benford, 198
- BenfordRV, 195
- Benford’s law
 - applications, 199–205
 - conditions for conformance to, 195–199
- Bernoulli trials, 128
- BernoulliRV, 114
- Beta conjugate distribution, 144
- Binary data, 141
- Binary trees, 102
- BingoCoverRV, 81
- BinomialRV, 89, 143
- Birnbaum
 - integrand g , 169
 - n -fold integral, 171
- BirthdayRV, 81
- Bootstrapping technique, 119, 141
 - algorithm for, 142–143
 - confidence interval, 143
- BootstrapRV, 146–150
- Brute force method, 95, 100
- CDF polynomials, coefficients of, 177
- CDF, 4, 37–39, 44, 88, 154, 158, 161, 179, 190, 191
- Central limit theorem, 3, 199–200
- Change-of-variable technique, 45
- CHF, 39, 44
- Chi square random variable, 35, 50
- Coefficient of variation, 160
- Computational probability, definition of, 3
- Computer algebra system, 8–10
- Conceptual algorithm development for sums of random variables, 95–106
- Confidence interval estimation, 141, 150–151
- Continuous random variable
 - distributions plotting, 40–41
 - expected value calculation, 41–42
 - functional form changing, 37–40
 - functions, 33
 - order statistics calculation, 42–43
 - product of two, 55–68
 - algorithm, 65–67
 - extensions, 64–65
 - generalizations, 58–59
 - implementation in APPL, 58–60
 - theorem, 56–58
 - transformation of, 45–54

- Continuous random variables and data structures
 - definition in APPL, using seven random variables, 34–35
 - sums, 91
- Convert, 73–74
- Convolution, 65, 91–118, 140, 158, 190
- ConvolutionIID, 4, 44, 102, 113, 116, 148, 199–200
- Critical path, 186
- Cumulative distribution function (CDF), 4, 33, 72
- Cumulative hazard function (CHF), 33, 72
- Data structures and continuous random variables, 33–37
 - changing functional form, 37–40
 - distribution plotting, 40–41
 - expected value calculation, 41–42
 - functions, 33
 - order statistics calculation, 42–43
- Data types, in Maple language, 18–20, 28
- Difference, 158, 188, 191
- Discrete convolution formula, 91
- Discrete-event input modeling situations, 164
- Discrete random variables, 5, 91, 119–120
- Dot support format, 76–77
- Economics and survival analysis, 144
- Error checking
 - conditional statements, 26
 - conversion routines, 28
 - in procedures, 28
- ExpectedValue, 41, 44, 89
- ExponentialRV, 39, 41–43, 137, 151, 161–164, 199, 204, 205
- Formulaic PDF, 76
- fsolve, 20–21, 163
- GammaRV, 140, 161, 167
- GeometricRV, 90, 125, 129, 204
- GetCriticalities, 189
- Hazard function (HF), 33, 72, 85
- HF, 39, 44, 136
- HypergeometricRV, 81, 115
- Identically distributed (iid) random variables, 3, 7
- IDF, 38–39, 44, 138, 139, 150–151, 200, 204
- Infinite support, 77
- Insertion sort algorithm, 96
- Inverse distribution function (IDF), 33, 72
- Kaplan–Meier product-limit survivor function, 164–165
- Kolmogorov–Smirnov goodness-of-fit test
 - CDF of KS test statistic, 181–183
 - the distribution D_n , 169–181
- KSRV, 156, 176, 179, 181–183
- KS test statistic, 169
- Kurtosis, 41, 44, 62, 89, 201
- Lehmer algorithm, 153, 158
- Lehmer generators, 153–154
- Lexicographical combination of n items, sampling of, 122, 130
- “List-of-sublists” data structure, 7, 19, 33–34
- Logarithm law, 194
- LognormalRV, 62
- Many-to-1 transformations, 45, 46
- Maple language
 - algebraic equations, 20–22
 - calculus problems, 24–25
 - conditional statements, 26
 - continuous random variables and data types, 18–20, 28
 - functions, 17–18
 - graphing using, 22–23
 - loops, 26–27
 - numeric computations, 13–14
 - procedures, 27–29
 - symbolic expression manipulation, 16–17
 - variables, 15–16

Maple (software)

- code, 160
- computation, 128–129
- error function, 38
- eval** procedure, 85
- implementation, 109
- list **Sample**, 154
- for** loop, 26, 140
- plot, 22
- simplify**, 16, 25, 40, 110, 179
- sorting algorithm for polynomials, 97
- unassign command, 161

Maple **evalf**(%) statement, 136

Maximum, 44, 138, 191

Maximum heap data structure, 96

Mean, 41, 44, 62, 113, 129, 136, 137, 204

MGF, 41, 44, 107

Minimum, 44, 138

Minimum heap, 101

Mixture, 44, 198

MLENHPP, 166

MLE, 44, 151, 161–166

MLEWeibull, 163

MOM, 44, 167

Moment generating functions (MGFs),
94–95

Monte Carlo simulation, 3, 194

Moving heap method, 100

MP6Censor, 164

$N \times n$ matrices *F*, 176

NegativeBinomialRV, 89

NextCombination and

NextPermutation, 122, 130

Node–arc incidence matrix, 187, 189

NoDot case, 73

NoDot support format, 75, 77

Non-overlapping ordered pairs, 157

Normal PDF approximation, 114

NormalRV, 38, 41, 63, 200, 201, 203

Numeric computations, in Maple
language, 13–14

Numeric PDF, 75

One dimensional array *Probs*, 95–96

OrderStat, 5, 6, 42–44, 120–122,
124–126, 128–130, 136, 146,
204, 206

OrderStat algorithm, 120–122, 125,
128–130

Order statistics

- APPL code to compute the PDF, 122
- computation of PDF of *r*th, 122
- equally likely probabilities, 121–122
- exact value, 129
- extension, 130–131
- notation and taxonomy, 119–120
- sampling with replacement, 126–129
 - finite support, 126–128
 - infinite support, 128–129
- sampling without replacement,
121–125
 - finite support, 121–124
 - infinite support, 125
- string "wo", importance, 121

OSgraphs, 205

Pascal's triangle, 174

PDF, 5, 6, 39, 44, 83–83, 86–87, 90, 113,
116, 122, 125, 179

Periodic transformations, 49

Piecewise many-to-1 transformation,
45, 46

PlotDist, 4, 40, 43, 88, 179, 202, 204,
205

PlotEmpVsFittedCDF, 154, 161, 165

Poisson distribution with PDF, 72, 87

Poisson random variables with
parameters λ_1 and λ_2 , 110

PoissonRV, 110

Power function, 7–8

Pre-defined APPL random variable
GeometricRV(*p*), 90

Probabilities and normal PDF ap-
proximations for $\Pr(S = s)$,
115–116

Probability density function (PDF), 4,
33, 72

by discrete convolution formula, 110

function, of *D*, 158

independent random variables
convolution, 91

Poisson random variable, 92

of *r*th order statistic, computation of,
122, 126

Probability mass function, 72

ProbStorage array, 122–125

- Procedures, in Maple, 27–29
 - Product**, 4, 55–68, 143
 - ProductIID**, 7, 65
- Random variables
 - functional form, 81
 - sums distribution calculation, 91
 - support, 78
- RangeStat** algorithm, 130, 145, 149
- ReduceList**, 29, 49
- Resampling error, 143
- Sampling
 - with replacement, 6–7
 - without replacement, 5–6
- Series-parallel activity networks, 187
- Seriessystemboot**, 142–143
- set.seed** function, 146
- SF**, 7, 38, 39, 44, 135–136, 140, 156
- Shellsort insertion sort, 97
- Significance level, 7
- Skewness**, 41, 44, 62, 89, 160–161
- Sociology and survival analysis, 144
- S-Plus
 - code, 146–149
 - function call, 143
- Standard continuity correction, 113
- Standard errors, estimation of, 145–150
- L*-Statistic, 145
- Stochastic activity network
 - algorithm, 189–194
 - matrix representation of a network, 186–187
 - notations and assumptions, 185–186
 - parallel reduction, 188
 - series-parallel activity network, 187
 - series reconstruction, 189
 - series reduction, 188
- Stochastic simulation
 - input modeling, 159–167
 - Kolmogorov–Smirnov goodness-of-fit test
 - CDF of KS test statistic, 181–183
 - computation of the distribution of D_n , 169–181
 - tests of randomness
 - distance between consecutive random number pairs, 156–158
 - Kolmogorov–Smirnov goodness-of-fit test, 154–156
- String object, 20
- Survivor function (SF), 7, 33, 72
- Symbolic expression manipulation, in Maple language, 16–17
- Systems of components
 - lower confidence bound p_L on system reliability, 140–144
 - survival analysis, 144–151
 - system analysis
 - analysis of k -out-of- n system, 135
 - determining probability of mission success, 140
 - system failure of parallel system, 137–138
 - system failure time of three-component system, 138–140
- system reliability estimates, 142
- Three-component series system, 141
- Transform**, 9, 44, 45–54, 64, 143, 148, 158, 162, 198–200
- Transform algorithm, 45–54
- Transformations, 1-to-1, 45
- Transformation technique for random variables, 54–55
- Transformation theorem, 46–48
- Triangular PDF plot, 40
- TriangularRV**, 4, 40, 41, 199
- Truncate**, 44, 138, 198
- Two-dimensional conceptual array construction, 97–98
- UniformDiscreteRV**, 5, 6, 113, 122, 128
- UniformRV**, 4, 9, 37, 50, 52, 61, 64, 65, 138, 139, 154, 158, 199, 200
- U*-subintervals, 170
- Variables, in Maple language, 15–16
- Variance**, 41, 44, 62, 89, 113, 129, 136, 146, 148
- VerifyPDF**, 9, 35, 44, 90
- Weibull survivor function, 164–165
- WeibullRV**, 38, 136, 163
- while** loop, in Maple, 27
- Zipf random variable, 80
- ZipfRV**, 81

**Early Titles in the
INTERNATIONAL SERIES IN OPERATIONS RESEARCH &
MANAGEMENT SCIENCE**

Frederick S. Hillier, Series Editor, Stanford University

- Saigal/A MODERN APPROACH TO LINEAR PROGRAMMING
- Nagurney/PROJECTED DYNAMICAL SYSTEMS & VARIATIONAL INEQUALITIES WITH APPLICATIONS
- Padberg & Rijal/LOCATION, SCHEDULING, DESIGN AND INTEGER PROGRAMMING
- Vanderbei/LINEAR PROGRAMMING
- Jaiswal/MILITARY OPERATIONS RESEARCH
- Gal & Greenberg/ADVANCES IN SENSITIVITY ANALYSIS & PARAMETRIC PROGRAMMING
- Prabhu/FOUNDATIONS OF QUEUEING THEORY
- Fang, Rajasekera & Tsao/ENTROPY OPTIMIZATION & MATHEMATICAL PROGRAMMING
- Yu/OR IN THE AIRLINE INDUSTRY
- Ho & Tang/PRODUCT VARIETY MANAGEMENT
- El-Taha & Stidham/SAMPLE-PATH ANALYSIS OF QUEUEING SYSTEMS
- Miettinen/NONLINEAR MULTIOBJECTIVE OPTIMIZATION
- Chao & Huntington/DESIGNING COMPETITIVE ELECTRICITY MARKETS
- Weglarz/PROJECT SCHEDULING: RECENT TRENDS & RESULTS
- Sahin & Polatoglu/QUALITY, WARRANTY AND PREVENTIVE MAINTENANCE
- Tavares/ADVANCES MODELS FOR PROJECT MANAGEMENT
- Tayur, Ganeshan & Magazine/QUANTITATIVE MODELS FOR SUPPLY CHAIN MANAGEMENT
- Weyant, J./ENERGY AND ENVIRONMENTAL POLICY MODELING
- Shanthikumar, J.G. & Sumita, U./APPLIED PROBABILITY AND STOCHASTIC PROCESSES
- Liu, B. & Esogbue, A.O./DECISION CRITERIA AND OPTIMAL INVENTORY PROCESSES
- Gal, T., Stewart, T.J., Hanne, T./MULTICRITERIA DECISION MAKING: Advances in MCDM Models, Algorithms, Theory, and Applications
- Fox, B.L./STRATEGIES FOR QUASI-MONTE CARLO
- Hall, R.W./HANDBOOK OF TRANSPORTATION SCIENCE
- Grassman, W.K./COMPUTATIONAL PROBABILITY
- Pomerol, J-C. & Barba-Romero, S./MULTICRITERION DECISION IN MANAGEMENT
- Axsäter, S./INVENTORY CONTROL
- Wolkowicz, H., Saigal, R., & Vandenberghe, L./HANDBOOK OF SEMI-DEFINITE PROGRAMMING: Theory, Algorithms, and Applications
- Hobbs, B.F. & Meier, P./ENERGY DECISIONS AND THE ENVIRONMENT: A Guide to the Use of Multicriteria Methods
- Dar-El, E./HUMAN LEARNING: From Learning Curves to Learning Organizations
- Armstrong, J.S./PRINCIPLES OF FORECASTING: A Handbook for Researchers and Practitioners
- Balsamo, S., Personé, V., & Onvural, R./ANALYSIS OF QUEUEING NETWORKS WITH BLOCKING
- Bouyssou, D. et al./EVALUATION AND DECISION MODELS: A Critical Perspective
- Hanne, T./INTELLIGENT STRATEGIES FOR META MULTIPLE CRITERIA DECISION MAKING
- Saaty, T. & Vargas, L./MODELS, METHODS, CONCEPTS and APPLICATIONS OF THE ANALYTIC HIERARCHY PROCESS
- Chatterjee, K. & Samuelson, W./GAME THEORY AND BUSINESS APPLICATIONS
- Hobbs, B. et al./THE NEXT GENERATION OF ELECTRIC POWER UNIT COMMITMENT MODELS
- Vanderbei, R.J./LINEAR PROGRAMMING: Foundations and Extensions, 2nd Ed.
- Kimms, A./MATHEMATICAL PROGRAMMING AND FINANCIAL OBJECTIVES FOR SCHEDULING PROJECTS

Early Titles in the
INTERNATIONAL SERIES IN OPERATIONS RESEARCH &
MANAGEMENT SCIENCE

(Continued)

- Baptiste, P., Le Pape, C. & Nuijten, W./*CONSTRAINT-BASED SCHEDULING*
- Feinberg, E. & Schwartz, A./*HANDBOOK OF MARKOV DECISION PROCESSES: Methods and Applications*
- Ramík, J. & Vlach, M./*GENERALIZED CONCAVITY IN FUZZY OPTIMIZATION AND DECISION ANALYSIS*
- Song, J. & Yao, D./*SUPPLY CHAIN STRUCTURES: Coordination, Information and Optimization*
- Kozan, E. & Ohuchi, A./*OPERATIONS RESEARCH/ MANAGEMENT SCIENCE AT WORK*
- Bouyssou et al./*AIDING DECISIONS WITH MULTIPLE CRITERIA: Essays in Honor of Bernard Roy*
- Cox, Louis Anthony, Jr./*RISK ANALYSIS: Foundations, Models and Methods*
- Dror, M., L'Ecuyer, P. & Szidarovszky, F./*MODELING UNCERTAINTY: An Examination of Stochastic Theory, Methods, and Applications*
- Dokuchaev, N./*DYNAMIC PORTFOLIO STRATEGIES: Quantitative Methods and Empirical Rules for Incomplete Information*
- Sarker, R., Mohammadian, M. & Yao, X./*EVOLUTIONARY OPTIMIZATION*
- Demeulemeester, R. & Herroelen, W./*PROJECT SCHEDULING: A Research Handbook*
- Gazis, D.C./*TRAFFIC THEORY*
- Zhu/*QUANTITATIVE MODELS FOR PERFORMANCE EVALUATION AND BENCHMARKING*
- Ehrgott & Gandibleux/*MULTIPLE CRITERIA OPTIMIZATION: State of the Art Annotated Bibliographical Surveys*
- Bienstock/*Potential Function Methods for Approx. Solving Linear Programming Problems*
- Matsatsinis & Siskos/*INTELLIGENT SUPPORT SYSTEMS FOR MARKETING DECISIONS*
- Alpern & Gal/*THE THEORY OF SEARCH GAMES AND RENDEZVOUS*
- Hall/*HANDBOOK OF TRANSPORTATION SCIENCE - 2nd Ed.*
- Glover & Kochenberger/*HANDBOOK OF METAHEURISTICS*
- Graves & Ringuest/*MODELS AND METHODS FOR PROJECT SELECTION: Concepts from Management Science, Finance and Information Technology*
- Hassin & Haviv/*TO QUEUE OR NOT TO QUEUE: Equilibrium Behavior in Queueing Systems*
- Gershwin et al./*ANALYSIS & MODELING OF MANUFACTURING SYSTEMS*
- Maros/*COMPUTATIONAL TECHNIQUES OF THE SIMPLEX METHOD*
- Harrison, Lee & Neale/*THE PRACTICE OF SUPPLY CHAIN MANAGEMENT: Where Theory and Application Converge*
- Shanthikumar, Yao & Zijm/*STOCHASTIC MODELING AND OPTIMIZATION OF MANUFACTURING SYSTEMS AND SUPPLY CHAINS*
- Nabrzyski, Schopf & Węglarz/*GRID RESOURCE MANAGEMENT: State of the Art and Future Trends*
- Thissen & Herder/*CRITICAL INFRASTRUCTURES: State of the Art in Research and Application*
- Carlsson, Fedrizzi, & Fullér/*FUZZY LOGIC IN MANAGEMENT*
- Soyer, Mazzuchi & Singpurwalla/*MATHEMATICAL RELIABILITY: An Expository Perspective*
- Chakravarty & Eliashberg/*MANAGING BUSINESS INTERFACES: Marketing, Engineering, and Manufacturing Perspectives*
- Talluri & van Ryzin/*THE THEORY AND PRACTICE OF REVENUE MANAGEMENT*
- Kavadias & Loch/*PROJECT SELECTION UNDER UNCERTAINTY: Dynamically Allocating Resources to Maximize Value*
- Brandeau, Sainfort & Pierskalla/*OPERATIONS RESEARCH AND HEALTH CARE: A Handbook of Methods and Applications*

Early Titles in the
**INTERNATIONAL SERIES IN OPERATIONS RESEARCH &
MANAGEMENT SCIENCE**

(Continued)

Cooper, Seiford & Zhu/*HANDBOOK OF DATA ENVELOPMENT ANALYSIS: Models and Methods*

Luenberger/*LINEAR AND NONLINEAR PROGRAMMING, 2nd Ed.*

Sherbrooke/*OPTIMAL INVENTORY MODELING OF SYSTEMS: Multi-Echelon Techniques, Second
Edition*

Chu, Leung, Hui & Cheung/*4th PARTY CYBER LOGISTICS FOR AIR CARGO*

Simchi-Levi, Wu & Shen/*HANDBOOK OF QUANTITATIVE SUPPLY CHAIN ANALYSIS:
Modeling in the E-Business Era*

Gass & Assad/*AN ANNOTATED TIMELINE OF OPERATIONS RESEARCH: An Informal History*

Greenberg/*TUTORIALS ON EMERGING METHODOLOGIES AND APPLICATIONS IN
OPERATIONS RESEARCH*

Weber/*UNCERTAINTY IN THE ELECTRIC POWER INDUSTRY: Methods and Models for Decision
Support*

Figueira, Greco & Ehrgott/*MULTIPLE CRITERIA DECISION ANALYSIS: State of the Art Surveys*

Reveliotis/*REAL-TIME MANAGEMENT OF RESOURCE ALLOCATIONS SYSTEMS: A Discrete Event
Systems Approach*

Kall & Mayer/*STOCHASTIC LINEAR PROGRAMMING: Models, Theory, and Computation*

**** A list of the more recent publications in the series is at the front of the book ****