# Computational Statistics Handbook with MATLAB®

Wendy L. Martinez
Angel R. Martinez

CHAPMAN & HALL/CRC

Boca Raton London New York Washington, D.C.

**Library of Congress Cataloging-in-Publication Data**

Catalog record is available from the Library of Congress

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

The consent of CRC Press LLC does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press LLC for such copying.

Direct all inquiries to CRC Press LLC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431.

**Trademark Notice:** Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation, without intent to infringe.

**Visit the CRC Press Web site at www.crcpress.com**

© 2002 by Chapman & Hall/CRC

No claim to original U.S. Government works
International Standard Book Number 1-58488-229-8
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

*To*

*Edward J. Wegman*

*Teacher, Mentor and Friend*

# Table of Contents

Preface ..... xv

## Chapter 1
### Introduction
1.1 What Is Computational Statistics? ..... 1
1.2 An Overview of the Book ..... 2
Philosophy ..... 2
What Is Covered ..... 3
A Word About Notation ..... 5
1.3 MATLAB Code ..... 6
Computational Statistics Toolbox ..... 7
Internet Resources ..... 7
1.4 Further Reading ..... 8

## Chapter 2
### Probability Concepts
2.1 Introduction ..... 11
2.2 Probability ..... 12
Background ..... 12
Probability ..... 14
Axioms of Probability ..... 17
2.3 Conditional Probability and Independence ..... 17
Conditional Probability ..... 17
Independence ..... 18
Bayes Theorem ..... 19
2.4 Expectation ..... 21
Mean and Variance ..... 21
Skewness ..... 23
Kurtosis ..... 23
2.5 Common Distributions ..... 24
Binomial ..... 24
Poisson ..... 26
Uniform ..... 29
Normal ..... 31
Exponential ..... 33
Gamma ..... 36
Chi-Square ..... 37
Weibull ..... 38
Beta ..... 40

Multivariate Normal ..... 41
2.6 MATLAB Code ..... 45
2.7 Further Reading ..... 46
Exercises ..... 48

## Chapter 3
### Sampling Concepts
3.1 Introduction ..... 51
3.2 Sampling Terminology and Concepts ..... 51
Sample Mean and Sample Variance ..... 53
Sample Moments ..... 54
Covariance ..... 56
3.3 Sampling Distributions ..... 58
3.4 Parameter Estimation ..... 60
Bias ..... 61
Mean Squared Error ..... 61
Relative Efficiency ..... 62
Standard Error ..... 62
Maximum Likelihood Estimation ..... 63
Method of Moments ..... 66
3.5 Empirical Distribution Function ..... 68
Quantiles ..... 69
3.6 MATLAB Code ..... 73
3.7 Further Reading ..... 74
Exercises ..... 76

## Chapter 4
### Generating Random Variables
4.1 Introduction ..... 79
4.2 General Techniques for Generating Random Variables ..... 79
Uniform Random Numbers ..... 79
Inverse Transform Method ..... 82
Acceptance-Rejection Method ..... 85
4.3 Generating Continuous Random Variables ..... 89
Normal Distribution ..... 89
Exponential Distribution ..... 89
Gamma ..... 91
Chi-Square ..... 93
Beta ..... 95
Multivariate Normal ..... 96
Generating Variates on a Sphere ..... 99
4.4 Generating Discrete Random Variables ..... 100
Binomial ..... 100
Poisson ..... 102
Discrete Uniform ..... 104

4.5 MATLAB Code ..... 106
4.6 Further Reading ..... 107
Exercises ..... 109

**Chapter 5**
**Exploratory Data Analysis**
5.1 Introduction ..... 111
5.2 Exploring Univariate Data ..... 112
Histograms ..... 113
Stem-and-Leaf ..... 116
Quantile-Based Plots - Continuous Distributions ..... 119
Q-Q Plot ..... 119
Quantile Plots ..... 123
Quantile Plots - Discrete Distributions ..... 126
Poissonness Plot ..... 126
Binomialness Plot ..... 129
Box Plots ..... 132
5.3 Exploring Bivariate and Trivariate Data ..... 135
Scatterplots ..... 135
Surface Plots ..... 138
Contour Plots ..... 138
Bivariate Histogram ..... 141
3-D Scatterplot ..... 145
5.4 Exploring Multi-Dimensional Data ..... 147
Scatterplot Matrix ..... 147
Slices and Isosurfaces ..... 149
Star Plots ..... 155
Andrews Curves ..... 157
Parallel Coordinates ..... 162
Projection Pursuit ..... 168
Projection Pursuit Index ..... 171
Finding the Structure ..... 172
Structure Removal ..... 174
Grand Tour ..... 178
5.5 MATLAB Code ..... 183
5.6 Further Reading ..... 184
Exercises ..... 187

**Chapter 6**
**Monte Carlo Methods for Inferential Statistics**
6.1 Introduction ..... 191
6.2 Classical Inferential Statistics ..... 192
Hypothesis Testing ..... 192
Confidence Intervals ..... 201
6.3 Monte Carlo Methods for Inferential Statistics ..... 204

Basic Monte Carlo Procedure ..... 204
Monte Carlo Hypothesis Testing ..... 205
Monte Carlo Assessment of Hypothesis Testing ..... 210
6.4 Bootstrap Methods ..... 214
General Bootstrap Methodology ..... 214
Bootstrap Estimate of Standard Error ..... 216
Bootstrap Estimate of Bias ..... 219
Bootstrap Confidence Intervals ..... 220
Bootstrap Standard Confidence Interval ..... 220
Bootstrap-*t* Confidence Interval ..... 221
Bootstrap Percentile Interval ..... 224
6.5 MATLAB Code ..... 226
6.6 Further Reading ..... 227
Exercises ..... 228

**Chapter 7**
**Data Partitioning**
7.1 Introduction ..... 231
7.2 Cross-Validation ..... 232
7.3 Jackknife ..... 239
7.4 Better Bootstrap Confidence Intervals ..... 247
7.5 Jackknife-After-Bootstrap ..... 251
7.6 MATLAB Code ..... 253
7.7 Further Reading ..... 254
Exercises ..... 256

**Chapter 8**
**Probability Density Estimation**
8.1 Introduction ..... 259
8.2 Histograms ..... 261
1-D Histograms ..... 261
Multivariate Histograms ..... 267
Frequency Polygons ..... 269
Averaged Shifted Histograms ..... 274
8.3 Kernel Density Estimation ..... 280
Univariate Kernel Estimators ..... 280
Multivariate Kernel Estimators ..... 285
8.4 Finite Mixtures ..... 287
Univariate Finite Mixtures ..... 289
Visualizing Finite Mixtures ..... 291
Multivariate Finite Mixtures ..... 293
EM Algorithm for Estimating the Parameters ..... 296
Adaptive Mixtures ..... 301
8.5 Generating Random Variables ..... 306
8.6 MATLAB Code ..... 311

8.7 Further Reading ..... 311
Exercises ..... 314

## Chapter 9
### Statistical Pattern Recognition
9.1 Introduction ..... 317
9.2 Bayes Decision Theory ..... 319
Estimating Class-Conditional Probabilities: Parametric Method ... 321
Estimating Class-Conditional Probabilities: Nonparametric ..... 322
Bayes Decision Rule ..... 323
Likelihood Ratio Approach ..... 329
9.3 Evaluating the Classifier ..... 332
Independent Test Sample ..... 333
Cross-Validation ..... 335
Receiver Operating Characteristic (ROC) Curve ..... 337
9.4 Classification Trees ..... 342
Growing the Tree ..... 347
Pruning the Tree ..... 352
Choosing the Best Tree ..... 356
Selecting the Best Tree Using an Independent Test Sample ... 357
Selecting the Best Tree Using Cross-Validation ..... 361
9.5 Clustering ..... 367
Measures of Distance ..... 367
Hierarchical Clustering ..... 369
*K*-Means Clustering ..... 373
9.6 MATLAB Code ..... 376
9.7 Further Reading ..... 379
Exercises ..... 381

## Chapter 10
### Nonparametric Regression
10.1 Introduction ..... 385
10.2 Smoothing ..... 390
Loess ..... 391
Robust Loess Smoothing ..... 396
Upper and Lower Smooths ..... 400
10.3 Kernel Methods ..... 401
Nadaraya-Watson Estimator ..... 404
Local Linear Kernel Estimator ..... 405
10.4 Regression Trees ..... 407
Growing a Regression Tree ..... 410
Pruning a Regression Tree ..... 411
Selecting a Tree ..... 412
10.5 MATLAB Code ..... 419
10.6 Further Reading ..... 420

Exercises ..... 422

## Chapter 11
### Markov Chain Monte Carlo Methods
11.1 Introduction ..... 425
11.2 Background ..... 426
Bayesian Inference ..... 426
Monte Carlo Integration ..... 427
Markov Chains ..... 429
Analyzing the Output ..... 430
11.3 Metropolis-Hastings Algorithms ..... 430
Metropolis-Hastings Sampler ..... 431
Metropolis Sampler ..... 433
Independence Sampler ..... 438
Autoregressive Generating Density ..... 439
11.4 The Gibbs Sampler ..... 443
11.5 Convergence Monitoring ..... 452
Gelman and Rubin Method ..... 453
Raftery and Lewis Method ..... 458
11.6 MATLAB Code ..... 458
11.7 Further Reading ..... 461
Exercises ..... 462

## Chapter 12
### Spatial Statistics
12.1 Introduction ..... 465
What Is Spatial Statistics? ..... 465
Types of Spatial Data ..... 466
Spatial Point Patterns ..... 467
Complete Spatial Randomness ..... 469
12.2 Visualizing Spatial Point Processes ..... 471
12.3 Exploring First-order and Second-order Properties ..... 475
Estimating the Intensity ..... 475
Estimating the Spatial Dependence ..... 478
Nearest Neighbor Distances - $G$ and $F$ Distributions ..... 478
$K$-Function ..... 482
12.4 Modeling Spatial Point Processes ..... 485
Nearest Neighbor Distances ..... 486
$K$-Function ..... 490
12.5 Simulating Spatial Point Processes ..... 495
Homogeneous Poisson Process ..... 495
Binomial Process ..... 497
Poisson Cluster Process ..... 499
Inhibition Process ..... 501
Strauss Process ..... 504

12.6 MATLAB Code ..... 505
12.7 Further Reading ..... 507
Exercises ..... 508

## Appendix A
**Introduction to MATLAB**
A.1 What Is MATLAB? ..... 511
A.2 Getting Help in MATLAB ..... 512
A.3 File and Workspace Management ..... 512
A.4 Punctuation in MATLAB ..... 514
A.5 Arithmetic Operators ..... 514
A.6 Data Constructs in MATLAB ..... 516
Basic Data Constructs ..... 516
Building Arrays ..... 516
Cell Arrays ..... 517
A.7 Script Files and Functions ..... 518
A.8 Control Flow ..... 520
For Loop ..... 520
While Loop ..... 520
If-Else Statements ..... 521
Switch Statement ..... 521
A.9 Simple Plotting ..... 521
A.10 Contact Information ..... 524

## Appendix B
**Index of Notation** ..... 525

## Appendix C
**Projection Pursuit Indexes**
C.1 Indexes ..... 529
Friedman-Tukey Index ..... 529
Entropy Index ..... 530
Moment Index ..... 530
Distances ..... 531
C.2 MATLAB Source Code ..... 532

## Appendix D
**MATLAB Code**
D.1 Bootstrap Confidence Interval ..... 539
D.2 Adaptive Mixtures Density Estimation ..... 540
D.3 Classification Trees ..... 542
D.4 Regression Trees ..... 544

**Appendix E**
**MATLAB Statistics Toolbox** ..... 547

**Appendix F**
**Computational Statistics Toolbox** ..... 557

**Appendix G**
**Data Sets** ..... 563

References ..... 571

Index ..... 585

# *Preface*

Computational statistics is a fascinating and relatively new field within statistics. While much of classical statistics relies on parameterized functions and related assumptions, the computational statistics approach is to let the data tell the story. The advent of computers with their number-crunching capability, as well as their power to show on the screen two- and three-dimensional structures, has made computational statistics available for any data analyst to use.

Computational statistics has a lot to offer the researcher faced with a file full of numbers. The methods of computational statistics can provide assistance ranging from preliminary exploratory data analysis to sophisticated probability density estimation techniques, Monte Carlo methods, and powerful multi-dimensional visualization. All of this power and novel ways of looking at data are accessible to researchers in their daily data analysis tasks. One purpose of this book is to facilitate the exploration of these methods and approaches and to provide the tools to make of this, not just a theoretical exploration, but a practical one. The two main goals of this book are:

- To make computational statistics techniques available to a wide range of users, including engineers and scientists, and
- To promote the use of MATLAB® by statisticians and other data analysts.

MATLAB and Handle Graphics® are registered trademarks of The MathWorks, Inc.

There are wonderful books that cover many of the techniques in computational statistics and, in the course of this book, references will be made to many of them. However, there are very few books that have endeavored to forgo the theoretical underpinnings to present the methods and techniques in a manner immediately usable to the practitioner. The approach we take in this book is to make computational statistics accessible to a wide range of users and to provide an understanding of statistics from a computational point of view via algorithms applied to real applications.

This book is intended for researchers in engineering, statistics, psychology, biostatistics, data mining and any other discipline that must deal with the analysis of raw data. Students at the senior undergraduate level or beginning graduate level in statistics or engineering can use the book to supplement course material. Exercises are included with each chapter, making it suitable as a textbook for a course in computational statistics and data analysis. Scien-

tists who would like to know more about programming methods for analyzing data in MATLAB would also find it useful.

We assume that the reader has the following background:

- Calculus: Since this book is computational in nature, the reader needs only a rudimentary knowledge of calculus. Knowing the definition of a derivative and an integral is all that is required.
- Linear Algebra: Since MATLAB is an array-based computing language, we cast several of the algorithms in terms of matrix algebra. The reader should have a familiarity with the notation of linear algebra, array multiplication, inverses, determinants, an array transpose, etc.
- Probability and Statistics: We assume that the reader has had introductory probability and statistics courses. However, we provide a brief overview of the relevant topics for those who might need a refresher.

We list below some of the major features of the book.

- The focus is on implementation rather than theory, helping the reader understand the concepts without being burdened by the theory.
- References that explain the theory are provided at the end of each chapter. Thus, those readers who need the theoretical underpinnings will know where to find the information.
- Detailed step-by-step algorithms are provided to facilitate implementation in any computer programming language or appropriate software. This makes the book appropriate for computer users who do not know MATLAB.
- MATLAB code in the form of a Computational Statistics Toolbox is provided. These functions are available for download at:

  `http://www.infinityassociates.com`

  `http://lib.stat.cmu.edu`.

  Please review the `readme` file for installation instructions and information on any changes.
- Exercises are given at the end of each chapter. The reader is encouraged to go through these, because concepts are sometimes explored further in them. Exercises are computational in nature, which is in keeping with the philosophy of the book.
- Many data sets are included with the book, so the reader can apply the methods to real problems and verify the results shown in the book. The data can also be downloaded separately from the toolbox at `http://www.infinityassociates.com`. The data are pro-

vided in MATLAB binary files (`.mat`) as well as text, for those who want to use them with other software.

- Typing in all of the commands in the examples can be frustrating. So, MATLAB scripts containing the commands used in the examples are also available for download at

  `http://www.infinityassociates.com`.

- A brief introduction to MATLAB is provided in Appendix A. Most of the constructs and syntax that are needed to understand the programming contained in the book are explained.
- An index of notation is given in Appendix B. Definitions and page numbers are provided, so the user can find the corresponding explanation in the text.
- Where appropriate, we provide references to internet resources for computer code implementing the algorithms described in the chapter. These include code for MATLAB, S-plus, Fortran, etc.

We would like to acknowledge the invaluable help of the reviewers: Noel Cressie, James Gentle, Thomas Holland, Tom Lane, David Marchette, Christian Posse, Carey Priebe, Adrian Raftery, David Scott, Jeffrey Solka, and Clifton Sutton. Their many helpful comments made this book a much better product. Any shortcomings are the sole responsibility of the authors. We owe a special thanks to Jeffrey Solka for some programming assistance with finite mixtures. We greatly appreciate the help and patience of those at CRC Press: Bob Stern, Joanne Blake, and Evelyn Meany. We also thank Harris Quesnell and James Yanchak for their help with resolving font problems. Finally, we are indebted to Naomi Fernandes and Tom Lane at The MathWorks, Inc. for their special assistance with MATLAB.

### Disclaimers

1. Any MATLAB programs and data sets that are included with the book are provided in good faith. The authors, publishers or distributors do not guarantee their accuracy and are not responsible for the consequences of their use.

2. The views expressed in this book are those of the authors and do not necessarily represent the views of DoD or its components.

Wendy L. and Angel R. Martinez
August 2001

# Chapter 1

## *Introduction*

### 1.1 What Is Computational Statistics?

Obviously, computational statistics relates to the traditional discipline of statistics. So, before we define computational statistics proper, we need to get a handle on what we mean by the field of statistics. At a most basic level, statistics is concerned with the transformation of raw data into knowledge [Wegman, 1988].

When faced with an application requiring the analysis of raw data, any scientist must address questions such as:

- What data should be collected to answer the questions in the analysis?
- How much data should be collected?
- What conclusions can be drawn from the data?
- How far can those conclusions be trusted?

Statistics is concerned with the science of uncertainty and can help the scientist deal with these questions. Many classical methods (regression, hypothesis testing, parameter estimation, confidence intervals, etc.) of statistics developed over the last century are familiar to scientists and are widely used in many disciplines [Efron and Tibshirani, 1991].

Now, what do we mean by computational statistics? Here we again follow the definition given in Wegman [1988]. Wegman defines computational statistics as a collection of techniques that have a strong "focus on the exploitation of computing in the creation of new statistical methodology."

Many of these methodologies became feasible after the development of inexpensive computing hardware since the 1980's. This computing revolution has enabled scientists and engineers to store and process massive amounts of data. However, these data are typically collected without a clear idea of what they will be used for in a study. For instance, in the practice of data analysis today, we often collect the data and then we design a study to

gain some useful information from them. In contrast, the traditional approach has been to first design the study based on research questions and then collect the required data.

Because the storage and collection is so cheap, the data sets that analysts must deal with today tend to be very large and high-dimensional. It is in situations like these where many of the classical methods in statistics are inadequate. As examples of computational statistics methods, Wegman [1988] includes parallel coordinates for high dimensional data representation, nonparametric functional inference, and data set mapping where the analysis techniques are considered fixed.

Efron and Tibshirani [1991] refer to what we call computational statistics as *computer-intensive statistical methods*. They give the following as examples for these types of techniques: bootstrap methods, nonparametric regression, generalized additive models and classification and regression trees. They note that these methods differ from the classical methods in statistics because they substitute computer algorithms for the more traditional mathematical method of obtaining an answer. An important aspect of computational statistics is that the methods free the analyst from choosing methods mainly because of their mathematical tractability.

Volume 9 of the *Handbook of Statistics: Computational Statistics* [Rao, 1993] covers topics that illustrate the "... trend in modern statistics of basic methodology supported by the state-of-the-art computational and graphical facilities..." It includes chapters on computing, density estimation, Gibbs sampling, the bootstrap, the jackknife, nonparametric function estimation, statistical visualization, and others.

We mention the topics that can be considered part of computational statistics to help the reader understand the difference between these and the more traditional methods of statistics. Table 1.1 [Wegman, 1988] gives an excellent comparison of the two areas.

## 1.2 An Overview of the Book

### Philosophy

The focus of this book is on methods of computational statistics and how to implement them. We leave out much of the theory, so the reader can concentrate on how the techniques may be applied. In many texts and journal articles, the theory obscures implementation issues, contributing to a loss of interest on the part of those needing to apply the theory. The reader should not misunderstand, though; the methods presented in this book are built on solid mathematical foundations. Therefore, at the end of each chapter, we

TABLE 1.1

Comparison Between Traditional Statistics and Computational Statistics [Wegman, 1988]. Reprinted with permission from the *Journal of the Washington Academy of Sciences.*

| Traditional Statistics | Computational Statistics |
|---|---|
| Small to moderate sample size | Large to very large sample size |
| Independent, identically distributed data sets | Nonhomogeneous data sets |
| One or low dimensional | High dimensional |
| Manually computational | Computationally intensive |
| Mathematically tractable | Numerically tractable |
| Well focused questions | Imprecise questions |
| Strong unverifiable assumptions:<br>Relationships (linearity, additivity)<br>Error structures (normality) | Weak or no assumptions:<br>Relationships (nonlinearity)<br>Error structures (distribution free) |
| Statistical inference | Structural inference |
| Predominantly closed form algorithms | Iterative algorithms possible |
| Statistical optimality | Statistical robustness |

include a section containing references that explain the theoretical concepts associated with the methods covered in that chapter.

### What Is Covered

In this book, we cover some of the most commonly used techniques in computational statistics. While we cannot include all methods that might be a part of computational statistics, we try to present those that have been in use for several years.

Since the focus of this book is on the implementation of the methods, we include algorithmic descriptions of the procedures. We also provide examples that illustrate the use of the algorithms in data analysis. It is our hope that seeing how the techniques are implemented will help the reader understand the concepts and facilitate their use in data analysis.

Some background information is given in Chapters 2, 3, and 4 for those who might need a refresher in probability and statistics. In Chapter 2, we discuss some of the general concepts of probability theory, focusing on how they

will be used in later chapters of the book. Chapter 3 covers some of the basic ideas of statistics and sampling distributions. Since many of the methods in computational statistics are concerned with estimating distributions via simulation, this chapter is fundamental to the rest of the book. For the same reason, we present some techniques for generating random variables in Chapter 4.

Some of the methods in computational statistics enable the researcher to explore the data before other analyses are performed. These techniques are especially important with high dimensional data sets or when the questions to be answered using the data are not well focused. In Chapter 5, we present some graphical exploratory data analysis techniques that could fall into the category of traditional statistics (e.g., box plots, scatterplots). We include them in this text so statisticians can see how to implement them in MATLAB and to educate scientists and engineers as to their usage in exploratory data analysis. Other graphical methods in this chapter do fall into the category of computational statistics. Among these are isosurfaces, parallel coordinates, the grand tour and projection pursuit.

In Chapters 6 and 7, we present methods that come under the general heading of resampling. We first cover some of the general concepts in hypothesis testing and confidence intervals to help the reader better understand what follows. We then provide procedures for hypothesis testing using simulation, including a discussion on evaluating the performance of hypothesis tests. This is followed by the bootstrap method, where the data set is used as an estimate of the population and subsequent sampling is done from the sample. We show how to get bootstrap estimates of standard error, bias and confidence intervals. Chapter 7 continues with two closely related methods called jackknife and cross-validation.

One of the important applications of computational statistics is the estimation of probability density functions. Chapter 8 covers this topic, with an emphasis on the nonparametric approach. We show how to obtain estimates using probability density histograms, frequency polygons, averaged shifted histograms, kernel density estimates, finite mixtures and adaptive mixtures.

Chapter 9 uses some of the concepts from probability density estimation and cross-validation. In this chapter, we present some techniques for statistical pattern recognition. As before, we start with an introduction of the classical methods and then illustrate some of the techniques that can be considered part of computational statistics, such as classification trees and clustering.

In Chapter 10 we describe some of the algorithms for nonparametric regression and smoothing. One nonparametric technique is a tree-based method called regression trees. Another uses the kernel densities of Chapter 8. Finally, we discuss smoothing using loess and its variants.

An approach for simulating a distribution that has become widely used over the last several years is called Markov chain Monte Carlo. Chapter 11 covers this important topic and shows how it can be used to simulate a posterior distribution. Once we have the posterior distribution, we can use it to estimate statistics of interest (means, variances, etc.).

We conclude the book with a chapter on spatial statistics as a way of showing how some of the methods can be employed in the analysis of spatial data. We provide some background on the different types of spatial data analysis, but we concentrate on spatial point patterns only. We apply kernel density estimation, exploratory data analysis, and simulation-based hypothesis testing to the investigation of spatial point processes.

We also include several appendices to aid the reader. Appendix A contains a brief introduction to MATLAB, which should help readers understand the code in the examples and exercises. Appendix B is an index to notation, with definitions and references to where it is used in the text. Appendices C and D include some further information about projection pursuit and MATLAB source code that is too lengthy for the body of the text. In Appendices E and F, we provide a list of the functions that are contained in the MATLAB Statistics Toolbox and the Computational Statistics Toolbox, respectively. Finally, in Appendix G, we include a brief description of the data sets that are mentioned in the book.

### A Word About Notation

The explanation of the algorithms in computational statistics (and the understanding of them!) depends a lot on notation. In most instances, we follow the notation that is used in the literature for the corresponding method. Rather than try to have unique symbols throughout the book, we think it is more important to be faithful to the convention to facilitate understanding of the theory and to make it easier for readers to make the connection between the theory and the text. Because of this, the same symbols might be used in several places.

In general, we try to stay with the convention that random variables are capital letters, whereas small letters refer to realizations of random variables. For example, $X$ is a random variable, and $x$ is an observed value of that random variable. When we use the term *log*, we are referring to the natural logarithm.

A symbol that is in bold refers to an array. Arrays can be row vectors, column vectors or matrices. Typically, a matrix is represented by a bold capital letter such as $\mathbf{B}$, while a vector is denoted by a bold lowercase letter such as $\mathbf{b}$. When we are using explicit matrix notation, then we specify the dimensions of the arrays. Otherwise, we do not hold to the convention that a vector always has to be in a column format. For example, we might represent a vector of observed random variables as $(x_1, x_2, x_3)$ or a vector of parameters as $(\mu, \sigma)$.

## 1.3 MATLAB Code

Along with the algorithmic explanation of the procedures, we include MATLAB commands to show how they are implemented. Any MATLAB commands, functions or data sets are in courier bold font. For example, **`plot`** denotes the MATLAB plotting function. The commands that are in the examples can be typed in at the command line to execute the examples. However, we note that due to typesetting considerations, we often have to continue a MATLAB command using the continuation punctuation (`...`). However, users do not have to include that with their implementations of the algorithms. See Appendix A for more information on how this punctuation is used in MATLAB.

Since this is a book about computational statistics, we assume the reader has the MATLAB Statistics Toolbox. In Appendix E, we include a list of functions that are in the toolbox and try to note in the text what functions are part of the main MATLAB software package and what functions are available only in the Statistics Toolbox.

The choice of MATLAB for implementation of the methods is due to the following reasons:

- The commands, functions and arguments in MATLAB are not cryptic. It is important to have a programming language that is easy to understand and intuitive, since we include the programs to help teach the concepts.
- It is used extensively by scientists and engineers.
- Student versions are available.
- It is easy to write programs in MATLAB.
- The source code or M-files can be viewed, so users can learn about the algorithms and their implementation.
- User-written MATLAB programs are freely available.
- The graphics capabilities are excellent.

It is important to note that the MATLAB code given in the body of the book is for ***learning purposes***. In many cases, it is not the most efficient way to program the algorithm. One of the purposes of including the MATLAB code is to help the reader understand the algorithms, especially how to implement them. So, we try to have the code match the procedures and to stay away from cryptic programming constructs. For example, we use **`for`** loops at times (when unnecessary!) to match the procedure. We make no claims that our code is the best way or the only way to program the algorithms.

In some cases, the MATLAB code is contained in an appendix, rather than in the corresponding chapter. These are applications where the MATLAB

program does not provide insights about the algorithms. For example, with classification and regression trees, the code can be quite complicated in places, so the functions are relegated to an appendix (Appendix D). Including these in the body of the text would distract the reader from the important concepts being presented.

### Computational Statistics Toolbox

The majority of the algorithms covered in this book are not available in MATLAB. So, we provide functions that implement most of the procedures that are given in the text. Note that these functions are a little different from the MATLAB code provided in the examples. In most cases, the functions allow the user to implement the algorithms for the general case. A list of the functions and their purpose is given in Appendix F. We also give a summary of the appropriate functions at the end of each chapter.

The MATLAB functions for the book are part of what we are calling the Computational Statistics Toolbox. To make it easier to recognize these functions, we put the letters '`cs`' in front. The toolbox can be downloaded from

- `http://lib.stat.cmu.edu`
- `http://www.infinityassociates.com`

Information on installing the toolbox is given in the `readme` file and on the website.

### Internet Resources

One of the many strong points about MATLAB is the availability of functions written by users, most of which are freely available on the internet. With each chapter, we provide information about internet resources for MATLAB programs (and other languages) that pertain to the techniques covered in the chapter.

The following are some internet sources for MATLAB code. Note that these are not necessarily specific to statistics, but are for all areas of science and engineering.

- The main website at The MathWorks, Inc. has code written by users and technicians of the company. The website for user contributed M-files is:

  `http://www.mathworks.com/support/ftp/`

  The website for M-files contributed by The MathWorks, Inc. is:

  `ftp://ftp.mathworks.com/pub/mathworks/`
- Another excellent resource for MATLAB programs is

`http://www.mathtools.net`.

At this site, you can sign up to be notified of new submissions.

- The main website for user contributed statistics programs is StatLib at Carnegie Mellon University. They have a new section containing MATLAB code. The home page for StatLib is

  `http://lib.stat.cmu.edu`

- We also provide the following internet sites that contain a list of MATLAB code available for purchase or download.

  `http://dmoz.org/Science/Math/Software/MATLAB/`

  `http://directory.google.com/Top/Science/Math/Software/MATLAB/`

## 1.4 Further Reading

To gain more insight on what is computational statistics, we refer the reader to the seminal paper by Wegman [1988]. Wegman discusses many of the differences between traditional and computational statistics. He also includes a discussion on what a graduate curriculum in computational statistics should consist of and contrasts this with the more traditional course work. A later paper by Efron and Tibshirani [1991] presents a summary of the new focus in statistical data analysis that came about with the advent of the computer age. Other papers in this area include Hoaglin and Andrews [1975] and Efron [1979]. Hoaglin and Andrews discuss the connection between computing and statistical theory and the importance of properly reporting the results from simulation experiments. Efron's article presents a survey of computational statistics techniques (the jackknife, the bootstrap, error estimation in discriminant analysis, nonparametric methods, and more) for an audience with a mathematics background, but little knowledge of statistics. Chambers [1999] looks at the concepts underlying computing with data, including the challenges this presents and new directions for the future.

There are very few general books in the area of computational statistics. One is a compendium of articles edited by C. R. Rao [1993]. This is a fairly comprehensive overview of many topics pertaining to computational statistics. The new text by Gentle [2001] is an excellent resource in computational statistics for the student or researcher. A good reference for statistical computing is Thisted [1988].

For those who need a resource for learning MATLAB, we recommend a wonderful book by Hanselman and Littlefield [1998]. This gives a comprehensive overview of MATLAB Version 5 and has been updated for Version 6 [Hanselman and Littlefield, 2001]. These books have information about the many capabilities of MATLAB, how to write programs, graphics and GUIs,

and much more. For the beginning user of MATLAB, these are a good place to start.

# Chapter 2

## *Probability Concepts*

### 2.1 Introduction

A review of probability is covered here at the outset because it provides the foundation for what is to follow: computational statistics. Readers who understand probability concepts may safely skip over this chapter.

Probability is the mechanism by which we can manage the uncertainty that underlies all real world data and phenomena. It enables us to gauge our degree of belief and to quantify the lack of certitude that is inherent in the process that generates the data we are analyzing. For example:

- To understand and use statistical hypothesis testing, one needs knowledge of the sampling distribution of the test statistic.
- To evaluate the performance (e.g., standard error, bias, etc.) of an estimate, we must know its sampling distribution.
- To adequately simulate a real system, one needs to understand the probability distributions that correctly model the underlying processes.
- To build classifiers to predict what group an object belongs to based on a set of features, one can estimate the probability density function that describes the individual classes.

In this chapter, we provide a brief overview of probability concepts and distributions as they pertain to computational statistics. In Section 2.2, we define probability and discuss some of its properties. In Section 2.3, we cover conditional probability, independence and Bayes' Theorem. Expectations are defined in Section 2.4, and common distributions and their uses in modeling physical phenomena are discussed in Section 2.5. In Section 2.6, we summarize some MATLAB functions that implement the ideas from Chapter 2. Finally, in Section 2.7 we provide additional resources for the reader who requires a more theoretical treatment of probability.

## 2.2 Probability

### Background

A ***random experiment*** is defined as a process or action whose outcome cannot be predicted with certainty and would likely change when the experiment is repeated. The variability in the outcomes might arise from many sources: slight errors in measurements, choosing different objects for testing, etc. The ability to model and analyze the outcomes from experiments is at the heart of statistics. Some examples of random experiments that arise in different disciplines are given below.

- Engineering: Data are collected on the number of failures of piston rings in the legs of steam-driven compressors. Engineers would be interested in determining the probability of piston failure in each leg and whether the failure varies among the compressors [Hand, et al., 1994].
- Medicine: The oral glucose tolerance test is a diagnostic tool for early diabetes mellitus. The results of the test are subject to variation because of different rates at which people absorb the glucose, and the variation is particularly noticeable in pregnant women. Scientists would be interested in analyzing and modeling the variation of glucose before and after pregnancy [Andrews and Herzberg, 1985].
- Manufacturing: Manufacturers of cement are interested in the tensile strength of their product. The strength depends on many factors, one of which is the length of time the cement is dried. An experiment is conducted where different batches of cement are tested for tensile strength after different drying times. Engineers would like to determine the relationship between drying time and tensile strength of the cement [Hand, et al., 1994].
- Software Engineering: Engineers measure the failure times in CPU seconds of a command and control software system. These data are used to obtain models to predict the reliability of the software system [Hand, et al., 1994].

The ***sample space*** is the set of all outcomes from an experiment. It is possible sometimes to list all outcomes in the sample space. This is especially true in the case of some discrete random variables. Examples of these sample spaces are:

- When observing piston ring failures, the sample space is $\{1, 0\}$, where 1 represents a failure and 0 represents a non-failure.
- If we roll a six-sided die and count the number of dots on the face, then the sample space is $\{1, 2, 3, 4, 5, 6\}$.

The outcomes from random experiments are often represented by an uppercase variable such as $X$. This is called a ***random variable***, and its value is subject to the uncertainty intrinsic to the experiment. Formally, a random variable is a real-valued function defined on the sample space. As we see in the remainder of the text, a random variable can take on different values according to a probability distribution. Using our examples of experiments from above, a random variable $X$ might represent the failure time of a software system or the glucose level of a patient. The observed value of a random variable $X$ is denoted by a lowercase $x$. For instance, a random variable $X$ might represent the number of failures of piston rings in a compressor, and $x = 5$ would indicate that we observed 5 piston ring failures.

Random variables can be discrete or continuous. A ***discrete random variable*** can take on values from a finite or countably infinite set of numbers. Examples of discrete random variables are the number of defective parts or the number of typographical errors on a page. A ***continuous random variable*** is one that can take on values from an interval of real numbers. Examples of continuous random variables are the inter-arrival times of planes at a runway, the average weight of tablets in a pharmaceutical production line or the average voltage of a power plant at different times.

We cannot list all outcomes from an experiment when we observe a continuous random variable, because there are an infinite number of possibilities. However, we could specify the interval of values that $X$ can take on. For example, if the random variable $X$ represents the tensile strength of cement, then the sample space might be $(0, \infty)$ kg/cm$^2$.

An ***event*** is a subset of outcomes in the sample space. An event might be that a piston ring is defective or that the tensile strength of cement is in the range 40 to 50 kg/cm$^2$. The probability of an event is usually expressed using the random variable notation illustrated below.

- Discrete Random Variables: Letting 1 represent a defective piston ring and letting 0 represent a good piston ring, then the probability of the event that a piston ring is defective would be written as

$$P(X = 1).$$

- Continuous Random Variables: Let $X$ denote the tensile strength of cement. The probability that an observed tensile strength is in the range 40 to 50 kg/cm$^2$ is expressed as

$$P(40 \text{ kg/cm}^2 \leq X \leq 50 \text{ kg/cm}^2).$$

Some events have a special property when they are considered together. Two events that cannot occur simultaneously or jointly are called ***mutually exclusive events***. This means that the intersection of the two events is the empty set and the probability of the events occurring together is zero. For example, a piston ring cannot be both defective and good at the same time. So, the event of getting a defective part and the event of getting a good part are mutually exclusive events. The definition of mutually exclusive events can be extended to any number of events by considering all pairs of events. Every pair of events must be mutually exclusive for all of them to be mutually exclusive.

## Probability

***Probability*** is a measure of the likelihood that some event will occur. It is also a way to quantify or to gauge the likelihood that an observed measurement or random variable will take on values within some set or range of values. Probabilities always range between 0 and 1. A ***probability distribution*** of a random variable describes the probabilities associated with each possible value for the random variable.

We first briefly describe two somewhat classical methods for assigning probabilities: the ***equal likelihood model*** and the ***relative frequency method***. When we have an experiment where each of $n$ outcomes is equally likely, then we assign a probability mass of $1/n$ to each outcome. This is the equal likelihood model. Some experiments where this model can be used are flipping a fair coin, tossing an unloaded die or randomly selecting a card from a deck of cards.

When the equal likelihood assumption is not valid, then the relative frequency method can be used. With this technique, we conduct the experiment $n$ times and record the outcome. The probability of event $E$ is assigned by $P(E) = f/n$, where $f$ denotes the number of experimental outcomes that satisfy event $E$.

Another way to find the desired probability that an event occurs is to use a ***probability density function*** when we have continuous random variables or a ***probability mass function*** in the case of discrete random variables. Section 2.5 contains several examples of probability density (mass) functions. In this text, $f(x)$ is used to represent the probability mass or density function for either discrete or continuous random variables, respectively. We now discuss how to find probabilities using these functions, first for the continuous case and then for discrete random variables.

To find the probability that a continuous random variable falls in a particular interval of real numbers, we have to calculate the appropriate area under the curve of $f(x)$. Thus, we have to evaluate the integral of $f(x)$ over the interval of random variables corresponding to the event of interest. This is represented by

$$P(a \leq X \leq b) = \int_a^b f(x)dx. \tag{2.1}$$

The area under the curve of $f(x)$ between $a$ and $b$ represents the probability that an observed value of the random variable $X$ will assume a value between $a$ and $b$. This concept is illustrated in Figure 2.1 where the shaded area represents the desired probability.

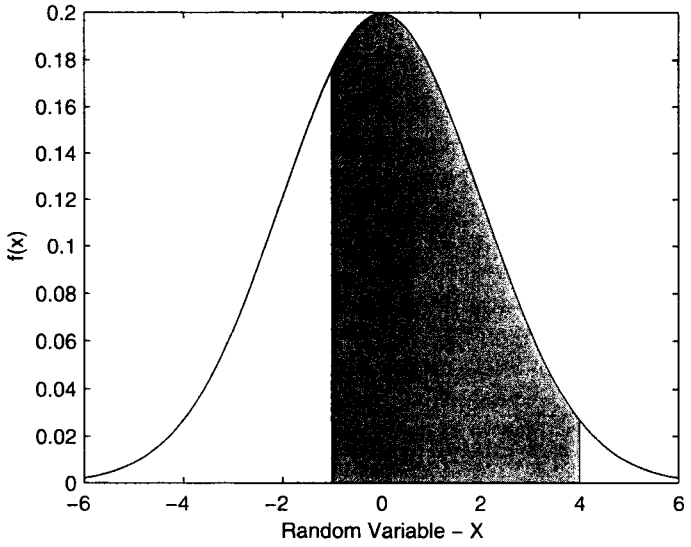



FIGURE 2.1
The area under the curve of $f(x)$ between -1 and 4 is the same as the probability that an observed value of the random variable will assume a value in the same interval.

It should be noted that a valid probability density function should be non-negative, and the total area under the curve must equal 1. If this is not the case, then the probabilities will not be properly restricted to the interval $[0, 1]$. This will be an important consideration in Chapter 8 where we discuss probability density estimation techniques.

The ***cumulative distribution function*** $F(x)$ is defined as the probability that the random variable $X$ assumes a value less than or equal to a given $x$. This is calculated from the probability density function, as follows

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)dt. \tag{2.2}$$

It is obvious that the cumulative distribution function takes on values between 0 and 1, so $0 \leq F(x) \leq 1$. A probability density function, along with its associated cumulative distribution function are illustrated in Figure 2.2.

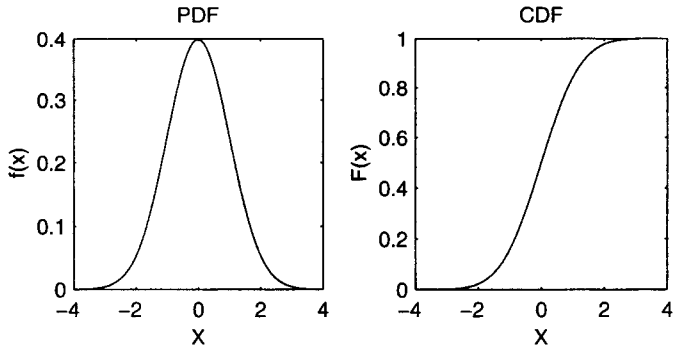



FIGURE 2.2
This shows the probability density function on the left with the associated cumulative distribution function on the right. Notice that the cumulative distribution function takes on values between 0 and 1.

For a discrete random variable $X$, that can take on values $x_1, x_2, \ldots$, the probability mass function is given by

$$f(x_i) = P(X = x_i); \qquad i = 1, 2, \ldots, \tag{2.3}$$

and the cumulative distribution function is

$$F(a) = \sum_{x_i \leq a} f(x_i); \qquad i = 1, 2, \ldots. \tag{2.4}$$

### Axioms of Probability

Probabilities follow certain axioms that can be useful in computational statistics. We let $S$ represent the sample space of an experiment and $E$ represent some event that is a subset of $S$.

*AXIOM 1*
*The probability of event E must be between 0 and 1:*

$$0 \leq P(E) \leq 1 .$$

*AXIOM 2*

$$P(S) = 1 .$$

*AXIOM 3*
*For mutually exclusive events,* $E_1, E_2, \ldots, E_k$,

$$P(E_1 \cup E_2 \cup \ldots \cup E_k) = \sum_{i=1}^{k} P(E_i) .$$

Axiom 1 has been discussed before and simply states that a probability must be between 0 and 1. Axiom 2 says that an outcome from our experiment must occur, and the probability that the outcome is in the sample space is 1. Axiom 3 enables us to calculate the probability that at least one of the mutually exclusive events $E_1, E_2, \ldots, E_k$ occurs by summing the individual probabilities.

## 2.3 Conditional Probability and Independence

### Conditional Probability

Conditional probability is an important concept. It is used to define independent events and enables us to revise our degree of belief given that another event has occurred. Conditional probability arises in situations where we need to calculate a probability based on some partial information concerning the experiment.

The ***conditional probability*** of event $E$ given event $F$ is defined as follows:

*CONDITIONAL PROBABILITY*

$$P(E|F) = \frac{P(E \cap F)}{P(F)}; \quad P(F) > 0. \tag{2.5}$$

Here $P(E \cap F)$ represents the ***joint probability*** that both $E$ and $F$ occur together and $P(F)$ is the probability that event $F$ occurs. We can rearrange Equation 2.5 to get the following rule:

*MULTIPLICATION RULE*

$$P(E \cap F) = P(F)P(E|F). \tag{2.6}$$

### Independence

Often we can assume that the occurrence of one event does not affect whether or not some other event happens. For example, say a couple would like to have two children, and their first child is a boy. The gender of their second child does not depend on the gender of the first child. Thus, the fact that we know they have a boy already does not change the probability that the second child is a boy. Similarly, we can sometimes assume that the value we observe for a random variable is not affected by the observed value of other random variables.

These types of events and random variables are called ***independent***. If *events* are independent, then knowing that one event has occurred does not change our degree of belief or the likelihood that the other event occurs. If *random variables* are independent, then the observed value of one random variable does not affect the observed value of another.

In general, the conditional probability $P(E|F)$ is not equal to $P(E)$. In these cases, the events are called ***dependent***. Sometimes we can assume independence based on the situation or the experiment, which was the case with our example above. However, to show independence mathematically, we must use the following definition.

*INDEPENDENT EVENTS*
*Two events E and F are said to be independent if and only if any of the following is true:*

$$\begin{aligned} P(E \cap F) &= P(E)P(F), \\ P(E) &= P(E|F). \end{aligned} \tag{2.7}$$

Note that if events $E$ and $F$ are independent, then the Multiplication Rule in Equation 2.6 becomes

$$P(E \cap F) = P(F)P(E),$$

which means that we simply multiply the individual probabilities for each event together. This can be extended to $k$ events to give

$$P(E_1 \cap E_2 \cap \ldots \cap E_k) = \prod_{i=1}^{k} P(E_i), \tag{2.8}$$

where events $E_i$ and $E_j$ (for all $i$ and $j$, $i \neq j$) are independent.

### Bayes Theorem

Sometimes we start an analysis with an initial degree of belief that an event will occur. Later on, we might obtain some additional information about the event that would change our belief about the probability that the event will occur. The initial probability is called a ***prior probability***. Using the new information, we can update the prior probability using Bayes' Theorem to obtain the ***posterior probability***.

The experiment of recording piston ring failure in compressors is an example of where Bayes' Theorem might be used, and we derive Bayes' Theorem using this example. Suppose our piston rings are purchased from two manufacturers: 60% from manufacturer A and 40% from manufacturer B.

Let $M_A$ denote the event that a part comes from manufacturer A, and $M_B$ represent the event that a piston ring comes from manufacturer B. If we select a part at random from our supply of piston rings, we would assign probabilities to these events as follows:

$$P(M_A) = 0.6,$$
$$P(M_B) = 0.4.$$

These are our prior probabilities that the piston rings are from the individual manufacturers.

Say we are interested in knowing the probability that a piston ring that subsequently failed came from manufacturer A. This would be the posterior probability that it came from manufacturer A, given that the piston ring failed. The additional information we have about the piston ring is that it failed, and we use this to update our degree of belief that it came from manufacturer A.

Bayes' Theorem can be derived from the definition of conditional probability (Equation 2.5). Writing this in terms of our events, we are interested in the following probability:

$$P(M_A|F) = \frac{P(M_A \cap F)}{P(F)}, \tag{2.9}$$

where $P(M_A|F)$ represents the posterior probability that the part came from manufacturer A, and $F$ is the event that the piston ring failed. Using the Multiplication Rule (Equation 2.6), we can write the numerator of Equation 2.9 in terms of event $F$ and our prior probability that the part came from manufacturer A, as follows

$$P(M_A|F) = \frac{P(M_A \cap F)}{P(F)} = \frac{P(M_A)P(F|M_A)}{P(F)}. \tag{2.10}$$

The next step is to find $P(F)$. The only way that a piston ring will fail is if: 1) it failed and it came from manufacturer A or 2) it failed and it came from manufacturer B. Thus, using the third axiom of probability, we can write

$$P(F) = P(M_A \cap F) + P(M_B \cap F).$$

Applying the Multiplication Rule as before, we have

$$P(F) = P(M_A)P(F|M_A) + P(M_B)P(F|M_B). \tag{2.11}$$

Substituting this for $P(F)$ in Equation 2.10, we write the posterior probability as

$$P(M_A|F) = \frac{P(M_A)P(F|M_A)}{P(M_A)P(F|M_A) + P(M_B)P(F|M_B)}. \tag{2.12}$$

Note that we need to find the probabilities $P(F|M_A)$ and $P(F|M_B)$. These are the probabilities that a piston ring will fail given it came from the corresponding manufacturer. These must be estimated in some way using available information (e.g., past failures). When we revisit Bayes' Theorem in the context of statistical pattern recognition (Chapter 9), these are the probabilities that are estimated to construct a certain type of classifier.

Equation 2.12 is Bayes' Theorem for a situation where only two outcomes are possible. In general, Bayes' Theorem can be written for any number of mutually exclusive events, $E_1, \ldots, E_k$, whose union makes up the entire sample space. This is given below.

*BAYES' THEOREM*

$$P(E_i|F) = \frac{P(E_i)P(F|E_i)}{P(E_1)P(F|E_1) + \ldots + P(E_k)P(F|E_k)}. \tag{2.13}$$

## 2.4 Expectation

Expected values and variances are important concepts in statistics. They are used to describe distributions, to evaluate the performance of estimators, to obtain test statistics in hypothesis testing, and many other applications.

### Mean and Variance

The ***mean*** or ***expected value*** of a random variable is defined using the probability density (mass) function. It provides a measure of central tendency of the distribution. If we observe many values of the random variable and take the average of them, we would expect that value to be close to the mean. The expected value is defined below for the discrete case.

*EXPECTED VALUE - DISCRETE RANDOM VARIABLES*

$$\mu = E[X] = \sum_{i=1}^{\infty} x_i f(x_i). \tag{2.14}$$

We see from the definition that the expected value is a sum of all possible values of the random variable where each one is weighted by the probability that X will take on that value.

The ***variance*** of a discrete random variable is given by the following definition.

*VARIANCE - DISCRETE RANDOM VARIABLES*
*For* $\mu < \infty$,

$$\sigma^2 = V(X) = E[(X-\mu)^2] = \sum_{i=1}^{\infty} (x_i-\mu)^2 f(x_i). \tag{2.15}$$

From Equation 2.15, we see that the variance is the sum of the squared distances, each one weighted by the probability that $X = x_i$. Variance is a measure of dispersion in the distribution. If a random variable has a large variance, then an observed value of the random variable is more likely to be far from the mean $\mu$. The standard deviation $\sigma$ is the square root of the variance.

The mean and variance for continuous random variables are defined similarly, with the summation replaced by an integral. The mean and variance of a continuous random variable are given below.

*EXPECTED VALUE - CONTINUOUS RANDOM VARIABLES*

$$\mu = E[X] = \int_{-\infty}^{\infty} xf(x)dx. \tag{2.16}$$

*VARIANCE - CONTINUOUS RANDOM VARIABLES*
*For* $\mu < \infty$,

$$\sigma^2 = V(X) = E[(X-\mu)^2] = \int_{-\infty}^{\infty} (x-\mu)^2 f(x)dx. \tag{2.17}$$

We note that Equation 2.17 can also be written as

$$V(X) = E[X^2] - \mu^2 = E[X^2] - (E[X])^2.$$

Other expected values that are of interest in statistics are the ***moments*** of a random variable. These are the expectation of powers of the random variable. In general, we define the ***r-th moment*** as

$$\mu'_r = E[X^r], \tag{2.18}$$

and the ***r-th central moment*** as

$$\mu_r = E[(X-\mu)^r]. \tag{2.19}$$

The mean corresponds to $\mu'_1$ and the variance is given by $\mu_2$.

### Skewness

The third central moment $\mu_3$ is often called a measure of asymmetry or skewness in the distribution. The uniform and the normal distribution are examples of symmetric distributions. The gamma and the exponential are examples of skewed or asymmetric distributions. The following ratio is called the ***coefficient of skewness***, which is often used to measure this characteristic:

$$\gamma_1 = \frac{\mu_3}{\mu_2^{3/2}}. \tag{2.20}$$

Distributions that are skewed to the left will have a negative coefficient of skewness, and distributions that are skewed to the right will have a positive value [Hogg and Craig, 1978]. The coefficient of skewness is zero for symmetric distributions. However, a coefficient of skewness equal to zero does not mean that the distribution must be symmetric.

### Kurtosis

Skewness is one way to measure a type of departure from normality. ***Kurtosis*** measures a different type of departure from normality by indicating the extent of the peak (or the degree of flatness near its center) in a distribution. The ***coefficient of kurtosis*** is given by the following ratio:

$$\gamma_2 = \frac{\mu_4}{\mu_2^2}. \tag{2.21}$$

We see that this is the ratio of the fourth central moment divided by the square of the variance. If the distribution is normal, then this ratio is equal to 3. A ratio greater than 3 indicates more values in the neighborhood of the mean (is more peaked than the normal distribution). If the ratio is less than 3, then it is an indication that the curve is flatter than the normal.

Sometimes the ***coefficient of excess kurtosis*** is used as a measure of kurtosis. This is given by

$$\gamma_2' = \frac{\mu_4}{\mu_2^2} - 3. \tag{2.22}$$

In this case, distributions that are more peaked than the normal correspond to a positive value of $\gamma_2'$, and those with a flatter top have a negative coefficient of excess kurtosis.

## 2.5 Common Distributions

In this section, we provide a review of some useful probability distributions and briefly describe some applications to modeling data. Most of these distributions are used in later chapters, so we take this opportunity to define them and to fix our notation. We first cover two important discrete distributions: the binomial and the Poisson. These are followed by several continuous distributions: the uniform, the normal, the exponential, the gamma, the chi-square, the Weibull, the beta and the multivariate normal.

### Binomial

Let's say that we have an experiment, whose outcome can be labeled as a 'success' or a 'failure'. If we let $X = 1$ denote a successful outcome and $X = 0$ represent a failure, then we can write the probability mass function as

$$\begin{aligned} f(0) &= P(X = 0) = 1 - p, \\ f(1) &= P(X = 1) = p, \end{aligned} \tag{2.23}$$

where $p$ represents the probability of a successful outcome. A random variable that follows the probability mass function in Equation 2.23 for $0 < p < 1$ is called a Bernoulli random variable.

Now suppose we repeat this experiment for $n$ trials, where each trial is independent (the outcome from one trial does not influence the outcome of another) and results in a success with probability $p$. If $X$ denotes the number of successes in these $n$ trials, then $X$ follows the binomial distribution with parameters $(n, p)$. Examples of binomial distributions with different parameters are shown in Figure 2.3.

To calculate a binomial probability, we use the following formula:

$$f(x;n,p) = P(X = x) = \binom{n}{x} p^x (1-p)^{n-x}; \qquad x = 0, 1, \ldots, n. \tag{2.24}$$

The mean and variance of a binomial distribution are given by

$$E[X] = np,$$

and

$$V(X) = np(1-p).$$

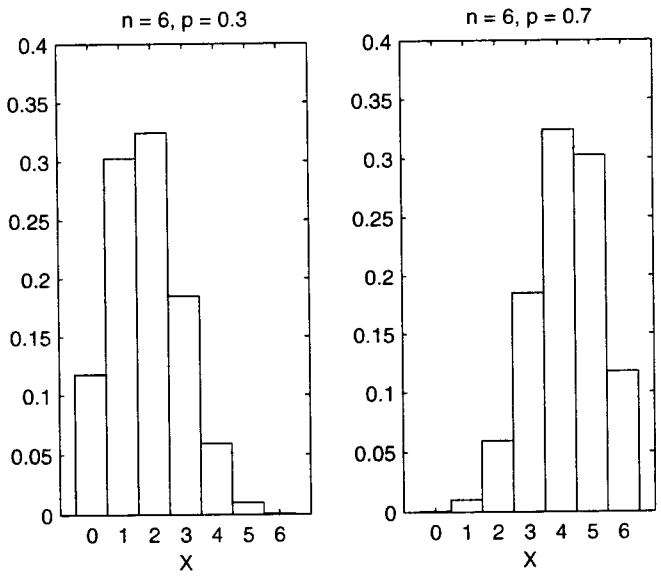



FIGURE 2.3
Examples of the binomial distribution for different success probabilities.

Some examples where the results of an experiment can be modeled by a binomial random variable are:

- A drug has probability 0.90 of curing a disease. It is administered to 100 patients, where the outcome for each patient is either cured or not cured. If $X$ is the number of patients cured, then $X$ is a binomial random variable with parameters (100, 0.90).
- The National Institute of Mental Health estimates that there is a 20% chance that an adult American suffers from a psychiatric disorder. Fifty adult Americans are randomly selected. If we let $X$ represent the number who have a psychiatric disorder, then $X$ takes on values according to the binomial distribution with parameters (50, 0.20).
- A manufacturer of computer chips finds that on the average 5% are defective. To monitor the manufacturing process, they take a random sample of size 75. If the sample contains more than five defective chips, then the process is stopped. The binomial distribution with parameters (75, 0.05) can be used to model the random variable $X$, where $X$ represents the number of defective chips.

### Example 2.1

Suppose there is a 20% chance that an adult American suffers from a psychiatric disorder. We randomly sample 25 adult Americans. If we let $X$ represent the number of people who have a psychiatric disorder, then $X$ is a binomial random variable with parameters $(25, 0.20)$. We are interested in the probability that at most 3 of the selected people have such a disorder. We can use the MATLAB Statistics Toolbox function **binocdf** to determine $P(X \leq 3)$, as follows:

```
prob = binocdf(3,25,0.2);
```

We could also sum up the individual values of the probability mass function from $X = 0$ to $X = 3$:

```
prob2 = sum(binopdf(0:3,25,0.2));
```

Both of these commands return a probability of 0.234. We now show how to generate the binomial distributions shown in Figure 2.3.

```
% Get the values for the domain, x.
x = 0:6;
% Get the values of the probability mass function.
% First for n = 6, p = 0.3:
pdf1 = binopdf(x,6,0.3);
% Now for n = 6, p = 0.7:
pdf2 = binopdf(x,6,0.7);
```

Now we have the values for the probability mass function (or the heights of the bars). The plots are obtained using the following code.

```
% Do the plots.
subplot(1,2,1),bar(x,pdf1,1,'w')
title(' n = 6, p = 0.3')
xlabel('X'),ylabel('f(X)')
axis square
subplot(1,2,2),bar(x,pdf2,1,'w')
title(' n = 6, p = 0.7')
xlabel('X'),ylabel('f(X)')
axis square
```

❑

#### Poisson

A random variable $X$ is a Poisson random variable with parameter $\lambda$, $\lambda > 0$, if it follows the probability mass function given by

$$f(x;\lambda) = P(X = x) = e^{-\lambda}\frac{\lambda^x}{x!}; \quad x = 0, 1, \ldots \tag{2.25}$$

The expected value and variance of a Poisson random variable are both $\lambda$, thus,

$$E[X] = \lambda,$$

and

$$V(X) = \lambda.$$

The Poisson distribution can be used in many applications. Examples of situations where a discrete random variable might follow a Poisson distribution are:

- the number of typographical errors on a page,
- the number of vacancies in a company during a month, or
- the number of defects in a length of wire.

The Poisson distribution is often used to approximate the binomial. When $n$ is large and $p$ is small (so $np$ is moderate), then the number of successes occurring can be approximated by the Poisson random variable with parameter $\lambda = np$.

The Poisson distribution is also appropriate for some applications where events occur at points in time or space. We see it used in this context in Chapter 12, where we look at modeling spatial point patterns. Some other examples include the arrival of jobs at a business, the arrival of aircraft on a runway, and the breakdown of machines at a manufacturing plant. The number of events in these applications can be described by a ***Poisson process***.

Let $N(t)$, $t \geq 0$, represent the number of events that occur in the time interval $[0, t]$. For each interval $[0, t]$, $N(t)$ is a random variable that can take on values $0, 1, 2, \ldots$. If the following conditions are satisfied, then the counting process $\{N(t), t \geq 0\}$ is said to be a Poisson process with mean rate $\lambda$ [Ross, 2000]:

1. $N(0) = 0$.
2. The process has independent increments.
3. The number $N(t)$ of events in an interval of length $t$ follows a Poisson distribution with mean $\lambda t$. Thus, for $s \geq 0$, $t \geq 0$,

$$P(N(t+s) - N(s) = k) = e^{-\lambda t}\frac{(\lambda t)^k}{k!}; \qquad k = 0, 1, \ldots. \tag{2.26}$$

From the third condition, we know that the process has stationary increments. This means that the distribution of the number of events in an interval depends only on the length of the interval and not on the starting point. The

second condition specifies that the number of events in one interval does not affect the number of events in other intervals. The first condition states that the counting starts at time $t = 0$. The expected value of $N(t)$ is given by

$$E[N(t)] = \lambda t .$$

### Example 2.2

In preparing this text, we executed the spell check command, and the editor reviewed the manuscript for typographical errors. In spite of this, some mistakes might be present. Assume that the number of typographical errors per page follows the Poisson distribution with parameter $\lambda = 0.25$. We calculate the probability that a page will have at least two errors as follows:

$$P(X \geq 2) = 1 - \{P(X = 0) + P(X = 1)\} = 1 - e^{-0.25} - e^{-0.25}0.25 \approx 0.0265 .$$

We can get this probability using the MATLAB Statistics Toolbox function **poisscdf**. Note that $P(X = 0) + P(X = 1)$ is the Poisson cumulative distribution function for $a = 1$ (see Equation 2.4), which is why we use **1** as the argument to **poisscdf**.

```
prob = 1-poisscdf(1,0.25);
```

❑

### Example 2.3

Suppose that accidents at a certain intersection occur in a manner that satisfies the conditions for a Poisson process with a rate of 2 per week ($\lambda = 2$). What is the probability that at most 3 accidents will occur during the next 2 weeks? Using Equation 2.26, we have

$$P(N(2) \leq 3) = \sum_{k=0}^{3} P(N(2) = k) .$$

Expanding this out yields

$$P(N(2) \leq 3) = e^{-4} + 4e^{-4} + \frac{4^2}{2!}e^{-4} + \frac{4^3}{3!}e^{-4} \approx 0.4335 .$$

As before, we can use the **poisscdf** function with parameter given by $\lambda t = 2 \cdot 2$.

```
prob = poisscdf(3,2*2);
```

❑

### Uniform

Perhaps one of the most important distributions is the uniform distribution for continuous random variables. One reason is that the uniform (0, 1) distribution is used as the basis for simulating most random variables as we discuss in Chapter 4.

A random variable that is uniformly distributed over the interval $(a, b)$ follows the probability density function given by

$$f(x;a,b) = \frac{1}{b-a}; \quad a < x < b. \tag{2.27}$$

The parameters for the uniform are the interval endpoints, $a$ and $b$. The mean and variance of a uniform random variable are given by

$$E[X] = \frac{a+b}{2},$$

and

$$V(X) = \frac{(b-a)^2}{12}.$$

The cumulative distribution function for a uniform random variable is

$$F(x) = \begin{cases} 0; & x \le a \\ \frac{x-a}{b-a}; & a < x < b \\ 1; & x \ge b. \end{cases} \tag{2.28}$$

**Example 2.4**

In this example, we illustrate the uniform probability density function over the interval (0, 10), along with the corresponding cumulative distribution function. The MATLAB Statistics Toolbox functions **unifpdf** and **unifcdf** are used to get the desired functions over the interval.

```
% First get the domain over which we will
% evaluate the functions.
x = -1:.1:11;
% Now get the probability density function
% values at x.
pdf = unifpdf(x,0,10);
% Now get the cdf.
cdf = unifcdf(x,0,10);
```

Plots of the functions are provided in Figure 2.4, where the probability density function is shown in the left plot and the cumulative distribution on the right. These plots are constructed using the following MATLAB commands.

```
% Do the plots.
subplot(1,2,1),plot(x,pdf)
title('PDF')
xlabel('X'),ylabel('f(X)')
axis([-1 11 0 0.2])
axis square
subplot(1,2,2),plot(x,cdf)
title('CDF')
xlabel('X'),ylabel('F(X)')
axis([-1 11 0 1.1])
axis square
```

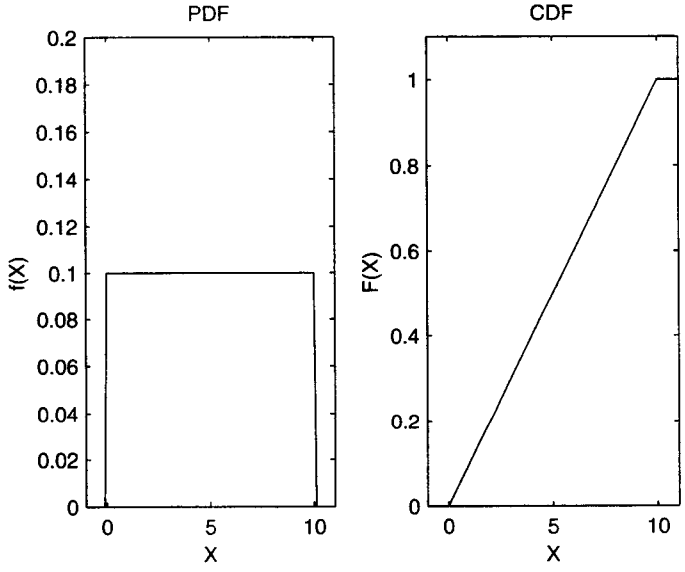



FIGURE 2.4
On the left is a plot of the probability density function for the uniform (0, 10). Note that the height of the curve is given by $1/(b-a) = 1/10 = 0.10$. The corresponding cumulative distribution function is shown on the right.

### Normal

A well known distribution in statistics and engineering is the normal distribution. Also called the Gaussian distribution, it has a continuous probability density function given by

$$f(x;\mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x-\mu)^2}{2\sigma^2}\right\}, \tag{2.29}$$

where $-\infty < x < \infty$; $-\infty < \mu < \infty$; $\sigma^2 > 0$. The normal distribution is completely determined by its parameters ($\mu$ and $\sigma^2$), which are also the expected value and variance for a normal random variable. The notation $X \sim N(\mu, \sigma^2)$ is used to indicate that a random variable $X$ is normally distributed with mean $\mu$ and variance $\sigma^2$. Several normal distributions with different parameters are shown in Figure 2.5.

Some special properties of the normal distribution are given here.

- The value of the probability density function approaches zero as $x$ approaches positive and negative infinity.
- The probability density function is centered at the mean $\mu$, and the maximum value of the function occurs at $x = \mu$.
- The probability density function for the normal distribution is symmetric about the mean $\mu$.

The special case of a standard normal random variable is one whose mean is zero ($\mu = 0$), and whose standard deviation is one ($\sigma = 1$). If $X$ is normally distributed, then

$$Z = \frac{X - \mu}{\sigma} \tag{2.30}$$

is a standard normal random variable.

Traditionally, the cumulative distribution function of a standard normal random variable is denoted by

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \exp\left\{-\frac{y^2}{2}\right\} dy. \tag{2.31}$$

The cumulative distribution function for a standard normal random variable can be calculated using the error function, denoted by *erf*. The relationship between these functions is given by

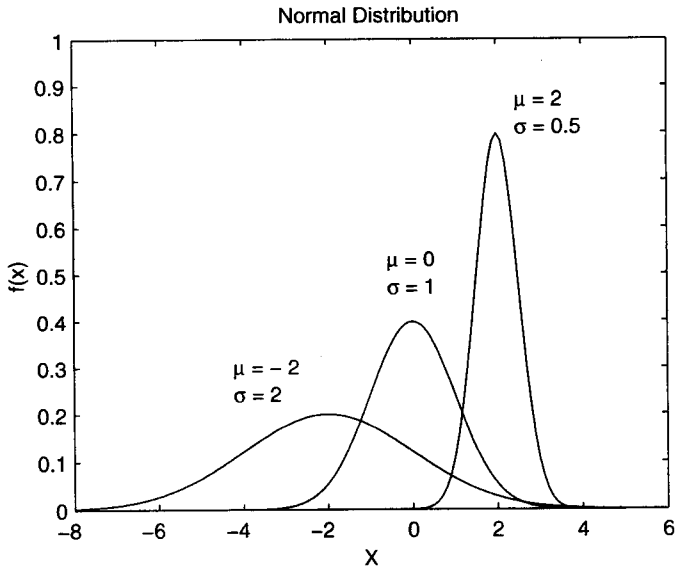



FIGURE 2.5
Examples of probability density functions for normally distributed random variables. Note that as the variance increases, the height of the probability density function at the mean decreases.

$$\Phi(z) = \frac{1}{2} erf\left(\frac{z}{\sqrt{2}}\right) + \frac{1}{2}. \tag{2.32}$$

The error function can be calculated in MATLAB using `erf(x)`. The MATLAB Statistics Toolbox has a function called `normcdf(x,mu,sigma)` that will calculate the cumulative distribution function for values in $x$. Its use is illustrated in the example given below.

## Example 2.5

Similar to the uniform distribution, the functions `normpdf` and `normcdf` are available in the MATLAB Statistics Toolbox for calculating the probability density function and cumulative distribution function for the normal. There is another special function called `normspec` that determines the probability that a random variable $X$ assumes a value between two limits, where $X$ is normally distributed with mean $\mu$ and standard deviation $\sigma$. This function also plots the normal density, where the area between the specified limits is shaded. The syntax is shown below.

```
% Set up the parameters for the normal distribution.
mu = 5;
sigma = 2;
% Set up the upper and lower limits. These are in
% the two element vector 'specs'.
specs = [2, 8];
prob = normspec(specs, mu, sigma);
```

The resulting plot is shown in Figure 2.6. Note that the default title and axes labels are shown, but these can be changed easily using the **title, xlabel,** and **ylabel** functions. You can also obtain tail probabilities by using **-Inf** as the first element of **specs** to designate no lower limit or **Inf** as the second element to indicate no upper limit.
□

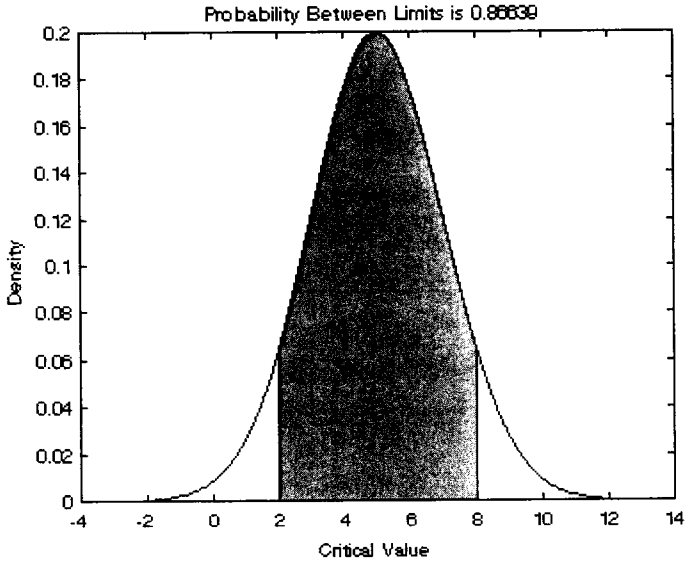



FIGURE 2.6
This shows the output from the function **normspec**. Note that it shades the area between the lower and upper limits that are specified as input arguments.

### Exponential

The exponential distribution can be used to model the amount of time until a specific event occurs or to model the time between independent events. Some examples where an exponential distribution could be used as the model are:

- the time until the computer locks up,
- the time between arrivals of telephone calls, or
- the time until a part fails.

The exponential probability density function with parameter $\lambda$ is

$$f(x;\lambda) = \lambda e^{-\lambda x}; \quad x \geq 0; \quad \lambda > 0. \tag{2.33}$$

The mean and variance of an exponential random variable are given by the following:

$$E[X] = \frac{1}{\lambda},$$

and

$$V(X) = \frac{1}{\lambda^2}.$$

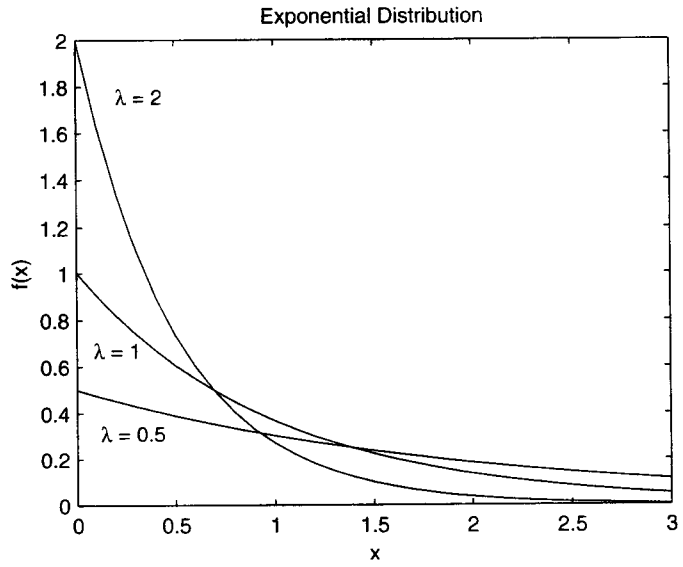



FIGURE 2.7
Exponential probability density functions for various values of $\lambda$.

The cumulative distribution function of an exponential random variable is given by

$$F(x) = \begin{cases} 0; & x < 0 \\ 1 - e^{-\lambda x}; & x \geq 0. \end{cases} \tag{2.34}$$

The exponential distribution is the only continuous distribution that has the memoryless property. This property describes the fact that the remaining lifetime of an object (whose lifetime follows an exponential distribution) does not depend on the amount of time it has already lived. This property is represented by the following equality, where $s \geq 0$ and $t \geq 0$:

$$P(X > s + t | X > s) = P(X > t).$$

In words, this means that the probability that the object will operate for time $s + t$, given it has already operated for time $s$, is simply the probability that it operates for time $t$.

When the exponential is used to represent interarrival times, then the parameter $\lambda$ is a rate with units of arrivals per time period. When the exponential is used to model the time until a failure occurs, then $\lambda$ is the failure rate. Several examples of the exponential distribution are shown in Figure 2.7.

### Example 2.6

The time between arrivals of vehicles at an intersection follows an exponential distribution with a mean of 12 seconds. What is the probability that the time between arrivals is 10 seconds or less? We are given the average interarrival time, so $\lambda = 1/12$. The required probability is obtained from Equation 2.34 as follows

$$P(X \leq 10) = 1 - e^{-(1/12)10} \approx 0.57.$$

You can calculate this using the MATLAB Statistics Toolbox function **`expocdf(x, 1/λ)`**. Note that this MATLAB function is based on a different definition of the exponential probability density function, which is given by

$$f(x;\mu) = \frac{1}{\mu} e^{-\frac{x}{\mu}}; \quad x \geq 0; \quad \mu > 0. \tag{2.35}$$

In the Computational Statistics Toolbox, we include a function called `csexpoc(x, λ)` that calculates the exponential cumulative distribution function using Equation 2.34.
❑

### Gamma

The gamma probability density function with parameters $\lambda > 0$ and $t > 0$ is

$$f(x;\lambda, t) = \frac{\lambda e^{-\lambda x}(\lambda x)^{t-1}}{\Gamma(t)}; \quad x \geq 0, \tag{2.36}$$

where $t$ is a shape parameter, and $\lambda$ is the scale parameter. The gamma function $\Gamma(t)$ is defined as

$$\Gamma(t) = \int_0^\infty e^{-y} y^{t-1} dy. \tag{2.37}$$

For integer values of $t$, Equation 2.37 becomes

$$\Gamma(t) = (t-1)!. \tag{2.38}$$

Note that for $t = 1$, the gamma density is the same as the exponential. When $t$ is a positive integer, the gamma distribution can be used to model the amount of time one has to wait until $t$ events have occurred, if the interarrival times are exponentially distributed.

The mean and variance of a gamma random variable are

$$E[X] = \frac{t}{\lambda},$$

and

$$V(X) = \frac{t}{\lambda^2}.$$

The cumulative distribution function for a gamma random variable is calculated using [Meeker and Escobar, 1998; Banks, et al., 2001]

$$F(x;\lambda, t) = \begin{cases} 0; & x \le 0 \\ \dfrac{1}{\Gamma(t)} \displaystyle\int_0^{\lambda x} y^{t-1} e^{-y} dy; & x > 0 . \end{cases} \tag{2.39}$$

Equation 2.39 can be evaluated easily in MATLAB using the **gammainc(λ*x,t)** function, where the above notation is used for the arguments.

### Example 2.7

We plot the gamma probability density function for $\lambda = t = 1$ (this should look like the exponential), $\lambda = t = 2$, and $\lambda = t = 3$. You can use the MATLAB Statistics Toolbox function **gampdf(x,t,1/λ)** or the function **csgammp(x,t,λ)**.

```
% First get the domain over which to
% evaluate the functions.
x = 0:.1:3;
% Now get the functions values for
% different values of lambda.
y1 = gampdf(x,1,1/1);
y2 = gampdf(x,2,1/2);
y3 = gampdf(x,3,1/3);
% Plot the functions.
plot(x,y1,'r',x,y2,'g',x,y3,'b')
title('Gamma Distribution')
xlabel('X')
ylabel('f(x)')
```

The resulting curves are shown in Figure 2.8.
❑

#### Chi-Square

A gamma distribution where $\lambda = 0.5$ and $t = \nu/2$, with $\nu$ a positive integer, is called a chi-square distribution (denoted as $\chi^2_\nu$) with $\nu$ degrees of freedom. The chi-square distribution is used to derive the distribution of the sample variance and is important for goodness-of-fit tests in statistical analysis [Mood, Graybill, and Boes, 1974].

The probability density function for a chi-square random variable with $\nu$ degrees of freedom is

$$f(x;\nu) = \frac{1}{\Gamma(\nu/2)} \left(\frac{1}{2}\right)^{\nu/2} x^{\nu/2-1} e^{-\frac{1}{2}x}; \qquad x \ge 0 . \tag{2.40}$$

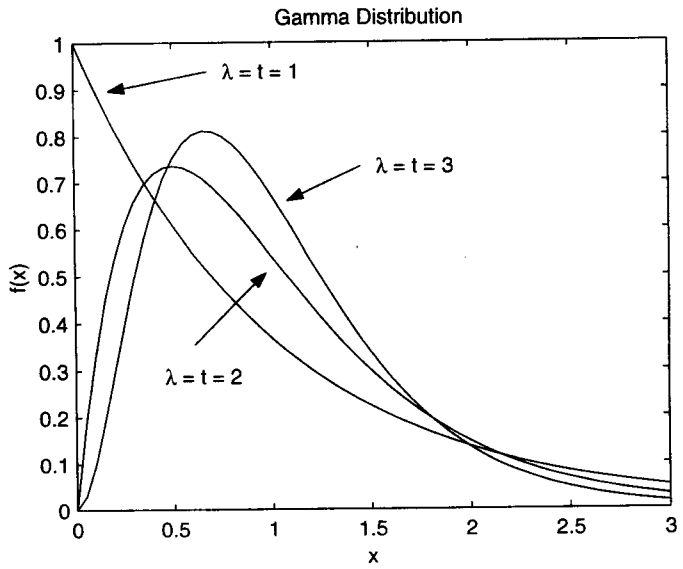



FIGURE 2.8
We show three examples of the gamma probability density function. We see that when $\lambda = t = 1$, we have the same probability density function as the exponential with parameter $\lambda = 1$.

The mean and variance of a chi-square random variable can be obtained from the gamma distribution. These are given by

$$E[X] = \nu,$$

and

$$V(X) = 2\nu.$$

### Weibull

The Weibull distribution has many applications in engineering. In particular, it is used in reliability analysis. It can be used to model the distribution of the amount of time it takes for objects to fail. For the special case where $\nu = 0$ and $\beta = 1$, the Weibull reduces to the exponential with $\lambda = 1/\alpha$.

The Weibull density for $\alpha > 0$ and $\beta > 0$ is given by

$$f(x;\nu,\alpha,\beta) = \left(\frac{\beta}{\alpha}\right)\left(\frac{x-\nu}{\alpha}\right)^{\beta-1} e^{-\left(\frac{x-\nu}{\alpha}\right)^{\beta}}; \qquad x > \nu, \tag{2.41}$$

and the cumulative distribution is

$$F(x;\nu,\alpha,\beta) = \begin{cases} 0; & x \le \nu \\ 1 - e^{-\left(\frac{x-\nu}{\alpha}\right)^{\beta}}; & x > \nu. \end{cases} \tag{2.42}$$

The location parameter is denoted by $\nu$, and the scale parameter is given by $\alpha$. The shape of the Weibull distribution is governed by the parameter $\beta$.

The mean and variance [Banks, et al., 2001] of a random variable from a Weibull distribution are given by

$$E[X] = \nu + \alpha\Gamma(1/\beta + 1),$$

and

$$V(X) = \alpha^2\left\{\Gamma(2/\beta + 1) - [\Gamma(1/\beta + 1)]^2\right\}.$$

### Example 2.8

Suppose the time to failure of piston rings for stream-driven compressors can be modeled by the Weibull distribution with a location parameter of zero, $\beta = 1/3$, and $\alpha = 500$. We can find the mean time to failure using the expected value of a Weibull random variable, as follows

$$E[X] = \nu + \alpha\Gamma(1/\beta + 1) = 500 \times \Gamma(3 + 1) = 3000 \text{ hours}.$$

Let's say we want to know the probability that a piston ring will fail before 2000 hours. We can calculate this probability using

$$F(2000;0, 500, 1/3) = 1 - \exp\left\{-\left(\frac{2000}{500}\right)^{1/3}\right\} \approx 0.796.$$

❑

You can use the MATLAB Statistics Toolbox function for applications where the location parameter is zero ($\nu = 0$). This function is called

**weibcdf** (for the cumulative distribution function), and the input arguments are: **(x,** $\alpha^{-\beta}$**,** $\beta$**)**. The reason for the different parameters is that MATLAB uses an alternate definition for the Weibull probability density function given by

$$f(x;a,b) = abx^{b-1}e^{-ax^b}; \qquad x > 0. \tag{2.43}$$

Comparing this with Equation 2.41, we can see that $\nu = 0$, $a = \alpha^{-\beta}$ and $b = \beta$. You can also use the function **csweibc(x,**$\nu$, $\alpha$, $\beta$**)** to evaluate the cumulative distribution function for a Weibull.

### Beta

The beta distribution is very flexible because it covers a range of different shapes depending on the values of the parameters. It can be used to model a random variable that takes on values over a bounded interval and assumes one of the shapes governed by the parameters. A random variable has a beta distribution with parameters $\alpha > 0$ and $\beta > 0$ if its probability density function is given by

$$f(x;\alpha,\beta) = \frac{1}{B(\alpha,\beta)}x^{\alpha-1}(1-x)^{\beta-1}; \qquad 0 < x < 1, \tag{2.44}$$

where

$$B(\alpha,\beta) = \int_0^1 x^{\alpha-1}(1-x)^{\beta-1}dx = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}. \tag{2.45}$$

The function $B(\alpha,\beta)$ can be calculated in MATLAB using the **beta(**$\alpha$,$\beta$**)** function. The mean and variance of a beta random variable are

$$E[X] = \frac{\alpha}{\alpha+\beta},$$

and

$$V(X) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}.$$

The cumulative distribution function for a beta random variable is given by integrating the beta probability density function as follows

$$F(x;\alpha,\beta) = \int_0^x \frac{1}{B(\alpha,\beta)} y^{\alpha-1}(1-y)^{\beta-1}dy. \tag{2.46}$$

The integral in Equation 2.46 is called the incomplete beta function. This can be calculated in MATLAB using the function **betainc(x,alpha,beta)**.

### EXAMPLE 2.9

We use the following MATLAB code to plot the beta density over the interval (0,1). We let $\alpha = \beta = 0.5$ and $\alpha = \beta = 3$.

```
% First get the domain over which to evaluate
% the density function.
x = 0.01:.01:.99;
% Now get the values for the density function.
y1 = betapdf(x,0.5,0.5);
y2 = betapdf(x,3,3);
% Plot the results.
plot(x,y1,'r',x,y2,'g')
title('Beta Distribution')
xlabel('x')
ylabel('f(x)')
```

The resulting curves are shown in Figure 2.9. You can use the MATLAB Statistics Toolbox function **betapdf(x,α,β)**, as we did in the example, or the function **csbetap(x,α,β)**.

❑

## Multivariate Normal

So far, we have discussed several univariate distributions for discrete and continuous random variables. In this section, we describe one of the important and most commonly used multivariate densities: the multivariate normal distribution. This distribution is used throughout the rest of the text. Some examples of where we use it are in exploratory data analysis, in probability density estimation, and in statistical pattern recognition.

The probability density function for a general multivariate normal density for $d$ dimensions is given by

$$f(\mathbf{x};\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}, \tag{2.47}$$

where $\mathbf{x}$ is a $d$-component column vector, $\boldsymbol{\mu}$ is the $d \times 1$ column vector of means, and $\boldsymbol{\Sigma}$ is the $d \times d$ covariance matrix. The superscript $T$ represents the

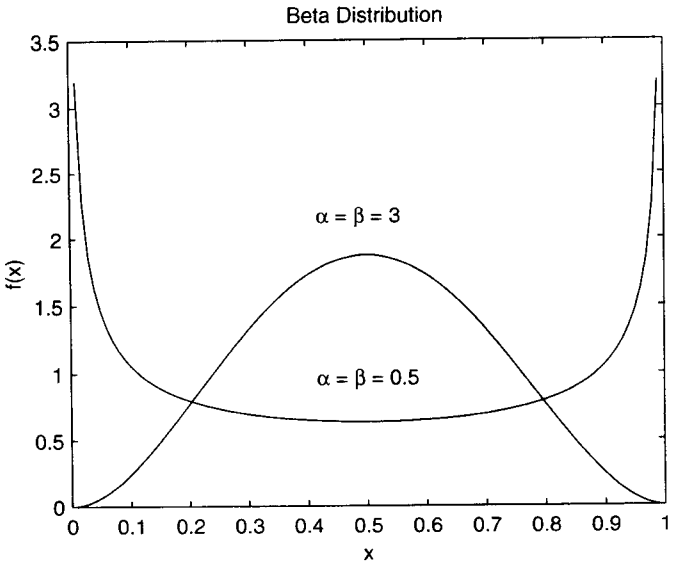



**FIGURE 2.9.**
Beta probability density functions for various parameters.

transpose of an array, and the notation $||$ denotes the determinant of a matrix.

The mean and covariance are calculated using the following formulas:

$$\boldsymbol{\mu} = E[\mathbf{x}], \tag{2.48}$$

and

$$\boldsymbol{\Sigma} = E[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T], \tag{2.49}$$

where the expected value of an array is given by the expected values of its components. Thus, if we let $X_i$ represent the $i$-th component of $\mathbf{x}$ and $\mu_i$ the $i$-th component of $\boldsymbol{\mu}$, then the elements of Equation 2.48 can be written as

$$\mu_i = E[X_i].$$

If $\sigma_{ij}$ represents the $ij$-th element of $\boldsymbol{\Sigma}$, then the elements of the covariance matrix (Equation 2.49) are given by

$$\sigma_{ij} = E[(X_i-\mu_i)(X_j-\mu_j)].$$

The covariance matrix is symmetric ($\mathbf{\Sigma}^T = \mathbf{\Sigma}$) positive definite (all eigenvalues of $\mathbf{\Sigma}$ are greater than zero) for most applications of interest to statisticians and engineers.

We illustrate some properties of the multivariate normal by looking at the bivariate ($d = 2$) case. The probability density function for a bivariate normal is represented by a bell-shaped surface. The center of the surface is determined by the mean $\boldsymbol{\mu}$ and the shape of the surface is determined by the covariance $\mathbf{\Sigma}$. If the covariance matrix is diagonal (all of the off-diagonal elements are zero), and the diagonal elements are equal, then the shape is circular. If the diagonal elements are not equal, then we get an ellipse with the major axis vertical or horizontal. If the covariance matrix is not diagonal, then the shape is elliptical with the axes at an angle. Some of these possibilities are illustrated in the next example.

### Example 2.10

We first provide the following MATLAB function to calculate the multivariate normal probability density function and illustrate its use in the bivariate case. The function is called **csevalnorm**, and it takes input arguments **x,mu,cov_mat**. The input argument **x** is a matrix containing the points in the domain where the function is to be evaluated, **mu** is a $d$-dimensional row vector, and **cov_mat** is the $d \times d$ covariance matrix.

```
function prob = csevalnorm(x,mu,cov_mat);
[n,d] = size(x);
% center the data points
x = x-ones(n,1)*mu;
a = (2*pi)^(d/2)*sqrt(det(cov_mat));
arg = diag(x*inv(cov_mat)*x');
prob = exp((-.5)*arg);
prob = prob/a;
```

We now call this function for a bivariate normal centered at zero and covariance matrix equal to the identity matrix. The density surface for this case is shown in Figure 2.10.

```
% Get the mean and covariance.
mu = zeros(1,2);
cov_mat = eye(2);% Identity matrix
% Get the domain.
% Should range (-4,4) in both directions.
[x,y] = meshgrid(-4:.2:4,-4:.2:4);
% Reshape into the proper format for the function.
X = [x(:),y(:)];
Z = csevalnorm(X,mu,cov_mat);
% Now reshape the matrix for plotting.
z = reshape(Z,size(x));
subplot(1,2,1) % plot the surface
```

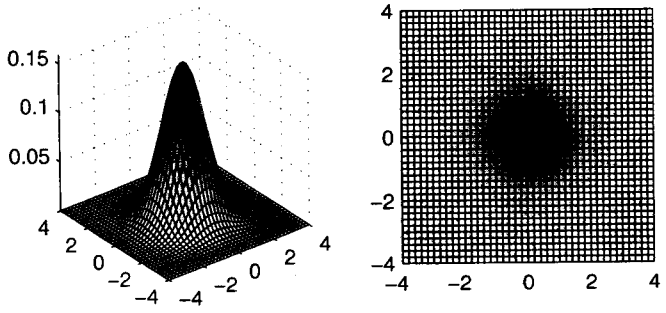



FIGURE 2.10
This figure shows a standard bivariate normal probability density function that is centered at the origin. The covariance matrix is given by the identity matrix. Notice that the shape of the surface looks circular. The plot on the right is for a viewpoint looking down on the surface.

```
surf(x,y,z),axis square, axis tight
title('BIVARIATE STANDARD NORMAL')
```

Next, we plot the surface for a bivariate normal centered at the origin with non-zero off-diagonal elements in the covariance matrix. Note the elliptical shape of the surface shown in Figure 2.11.

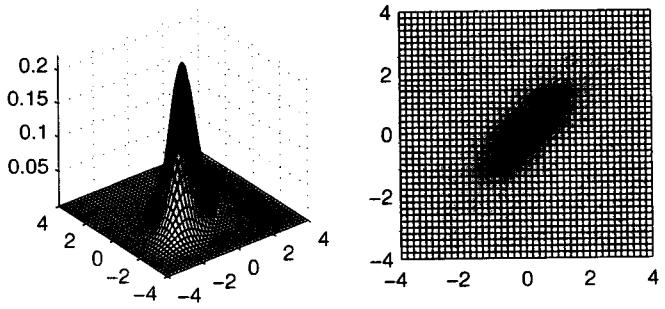



FIGURE 2.11
This shows a bivariate normal density where the covariance matrix has non-zero off-diagonal elements. Note that the surface has an elliptical shape. The plot on the right is for a viewpoint looking down on the surface.

```
subplot(1,2,2) % look down on the surface
pcolor(x,y,z),axis square
title('BIVARIATE STANDARD NORMAL')
% Now do the same thing for a covariance matrix
% with non-zero off-diagonal elements.
cov_mat = [1 0.7 ; 0.7 1];
Z = csevalnorm(X,mu,cov_mat);
z = reshape(Z,size(x));
subplot(1,2,1)
surf(x,y,z),axis square, axis tight
title('BIVARIATE NORMAL')
subplot(1,2,2)
pcolor(x,y,z),axis square
title('BIVARIATE NORMAL')
```

❑

The probability that a point $\mathbf{x} = (x_1, x_2)^T$ will assume a value in a region $R$ can be found by integrating the bivariate probability density function over the region. Any plane that cuts the surface parallel to the $x_1$-$x_2$ plane intersects in an elliptic (or circular) curve, yielding a curve of constant density. Any plane perpendicular to the $x_1$-$x_2$ plane cuts the surface in a normal curve. This property indicates that in each dimension, the multivariate normal is a univariate normal distribution. This is discussed further in Chapter 5.

## 2.6 MATLAB Code

The MATLAB Statistics Toolbox has many functions for the more common distributions. It has functions for finding the value of the probability density (mass) function and the value of the cumulative distribution function. The reader is cautioned to remember that the definitions of the distributions (exponential, gamma, and Weibull) differ from what we describe in the text. For example, the exponential and the gamma distributions are parameterized differently in the MATLAB Statistics Toolbox. For a complete list of what is available in the toolbox for calculating probability density (mass) functions or cumulative distribution functions, see Appendix E.

The Computational Statistics Toolbox contains functions for several of the distributions, as defined in this chapter. In general, those functions that end in `p` correspond to the probability density (mass) function, and those ending with a `c` calculate the cumulative distribution function. Table 2.1 provides a summary of the functions.

We note that a different function for evaluating the multivariate normal probability density function is available for download at

TABLE 2.1
List of Functions from Chapter 2 Included in the Computational Statistics Toolbox

| Distribution | MATLAB Function |
|---|---|
| Beta | `csbetap, csbetac` |
| Binomial | `csbinop, csbinoc` |
| Chi-square | `cschip, cschic` |
| Exponential | `csexpop, csexpoc` |
| Gamma | `csgammp, csgammc` |
| Normal - univariate | `csnormp, csnormc` |
| Normal - multivariate | `csevalnorm` |
| Poisson | `cspoisp, cspoisc` |
| Continuous Uniform | `csunifp, csunifc` |
| Weibull | `csweibp, csweibc` |

`ftp://ftp.mathworks.com/pub/mathworks/`

under the `stats` directory. This function can be substituted for `csevalnorm`.

## 2.7 Further Reading

There are many excellent books on probability theory at the undergraduate and graduate levels. Ross [1994; 1997; 2000] is the author of several books on probability theory and simulation. These texts contain many examples and are appropriate for advanced undergraduate students in statistics, engineering and science. Rohatgi [1976] provides a solid theoretical introduction to probability theory. This text can be used by advanced undergraduate and beginning graduate students. It has recently been updated with many new examples and special topics [Rohatgi and Saleh, 2000]. For those who want to learn about probability, but do not want to be overwhelmed with the theory, then we recommend Durrett [1994].

At the graduate level, there is a book by Billingsley [1995] on probability and measure theory. He uses probability to motivate measure theory and then uses measure theory to generate more probability concepts. Another good reference is a text on probability and real analysis by Ash [1972]. This is suitable for graduate students in mathematics and statistics. For a book that can be used by graduate students in mathematics, statistics and engineering, see Port [1994]. This text provides a comprehensive treatment of the subject and can also be used as a reference by professional data analysts. Finally, Breiman [1992] provides an overview of probability theory that is accessible to statisticians and engineers.

## Exercises

2.1. Write a function using MATLAB's functions for numerical integration such as **quad** or **quadl** (MATLAB 6) that will find $P(X \leq x)$ when the random variable is exponentially distributed with parameter $\lambda$. See **help** for information on how to use these functions.

2.2. Verify that the exponential probability density function with parameter $\lambda$ integrates to 1. Use the MATLAB functions **quad** or **quadl** (MATLAB 6). See **help** for information on how to use these functions.

2.3. Radar and missile detection systems warn of enemy attacks. Suppose that a radar detection system has a probability 0.95 of detecting a missile attack.

a. What is the probability that one detection system will detect an attack? What distribution did you use?

b. Suppose three detection systems are located together in the same area and the operation of each system is independent of the others. What is the probability that at least one of the systems will detect the attack? What distribution did you use in this case?

2.4. When a random variable is equally likely to be either positive or negative, then the Laplacian or the double exponential distribution can be used to model it. The Laplacian probability density function for $\lambda > 0$ is given by

$$f(x) = \frac{1}{2}\lambda e^{-\lambda|x|}; \quad -\infty < x < \infty.$$

a. Derive the cumulative distribution function for the Laplacian.

b. Write a MATLAB function that will evaluate the Laplacian probability density function for given values in the domain.

c. Write a MATLAB function that will evaluate the Laplacian cumulative distribution function.

d. Plot the probability density function when $\lambda = 1$.

2.5. Suppose $X$ follows the exponential distribution with parameter $\lambda$. Show that for $s \geq 0$ and $t \geq 0$,

$$P(X > s + t | X > s) = P(X > t).$$

2.6. The lifetime in years of a flat panel display is a random variable with the exponential probability density function given by

$$f(x;0.1) = 0.1e^{-0.1x}.$$

a. What is the mean lifetime of the flat panel display?

b. What is the probability that the display fails within the first two years?

c. Given that the display has been operating for one year, what is the probability that it will fail within the next year?

2.7. The time to failure for a widget follows a Weibull distribution, with $\nu = 0$, $\beta = 1/2$, and $\alpha = 750$ hours.

a. What is the mean time to failure of the widget?

b. What percentage of the widgets will fail by 2500 hours of operation? That is, what is the probability that a widget will fail within 2500 hours?

2.8. Let's say the probability of having a boy is 0.52. Using the Multiplication Rule, find the probability that a family's first and second children are boys. What is the probability that the first child is a boy and the second child is a girl?

2.9. Repeat Example 2.1 for $n = 6$ and $p = 0.5$. What is the shape of the distribution?

2.10. Recall that in our piston ring example, $P(M_A) = 0.6$ and $P(M_B) = 0.4$. From prior experience with the two manufacturers, we know that 2% of the parts supplied by manufacturer A are likely to fail and 6% of the parts supplied by manufacturer B are likely to fail. Thus, $P(F|M_A) = 0.02$ and $P(F|M_B) = 0.06$. If we observe a piston ring failure, what is the probability that it came from manufacturer A?

2.11. Using the functions **fminbnd** or **fmin** (available in the standard MATLAB package), find the value for $x$ where the maximum of the $N(3, 1)$ probability density occurs. Note that you have to find the minimum of $-f(x)$ to find the maximum of $f(x)$ using these functions. Refer to the **help** files on these functions for more information on how to use them.

2.12. Using **normpdf** or **csnormp**, find the value of the probability density for $N(0, 1)$ at $\pm\infty$. Use a small (large) value of $x$ for $-\infty$ ($\infty$).

2.13. Verify Equation 2.38 using the MATLAB functions **factorial** and **gamma**.

2.14. Find the height of the curve for a normal probability density function at $x = \mu$, where $\sigma = 0.5, 1, 2$. What happens to the height of the curve as $\sigma$ gets larger? Does the height change for different values of $\mu$?

2.15. Write a function that calculates the Bayes' posterior probability given a vector of conditional probabilities and a vector of prior probabilities.

2.16. Compare the Poisson approximation to the actual binomial probability $P(X = 4)$, using $n = 9$ and $p = 0.1, 0.2, \ldots, 0.9$.

2.17. Using the function **normspec**, find the probability that the random variable defined in Example 2.5 assumes a value that is less than 3. What is the probability that the same random variable assumes a value that is greater than 5? Find these probabilities again using the function **normcdf**.

2.18. Find the probability for the Weibull random variable of Example 2.8 using the MATLAB Statistics Toolbox function **weibcdf** or the Computational Statistics Toolbox function **csweibc**.

2.19. The MATLAB Statistics Toolbox has a GUI demo called **disttool**. First view the **help** file on **disttool**. Then run the demo. Examine the probability density (mass) and cumulative distribution functions for the distributions discussed in the chapter.

# Chapter 3

# *Sampling Concepts*

## 3.1 Introduction

In this chapter, we cover the concepts associated with random sampling and the sampling distribution of statistics. These notions are fundamental to computational statistics and are needed to understand the topics covered in the rest of the book. As with Chapter 2, those readers who have a basic understanding of these ideas may safely move on to more advanced topics.

In Section 3.2, we discuss the terminology and concepts associated with random sampling and sampling distributions. Section 3.3 contains a brief discussion of the Central Limit Theorem. In Section 3.4, we describe some methods for deriving estimators (maximum likelihood and the method of moments) and introduce criteria for evaluating their performance. Section 3.5 covers the empirical distribution function and how it is used to estimate quantiles. Finally, we conclude with a section on the MATLAB functions that are available for calculating the statistics described in this chapter and a section on further readings.

## 3.2 Sampling Terminology and Concepts

In Chapter 2, we introduced the idea of a random experiment. We typically perform an experiment where we collect data that will provide information on the phenomena of interest. Using these data, we draw conclusions that are usually beyond the scope of our particular experiment. The researcher generalizes from that experiment to the class of all similar experiments. This is the heart of inferential statistics. The problem with this sort of generalization is that we cannot be absolutely certain about our conclusions. However, by

using statistical techniques, we can measure and manage the degree of uncertainty in our results.

***Inferential statistics*** is a collection of techniques and methods that enable researchers to observe a subset of the objects of interest and using the information obtained from these observations make statements or inferences about the entire population of objects. Some of these methods include the estimation of population parameters, statistical hypothesis testing, and probability density estimation.

The ***target population*** is defined as the entire collection of objects or individuals about which we need some information. The target population must be well defined in terms of what constitutes membership in the population (e.g., income level, geographic area, etc.) and what characteristics of the population we are measuring (e.g., height, IQ, number of failures, etc.).

The following are some examples of populations, where we refer back to those described at the beginning of Chapter 2.

- For the piston ring example, our population is all piston rings contained in the legs of steam-driven compressors. We would be observing the time to failure for each piston ring.
- In the glucose example, our population might be all pregnant women, and we would be measuring the glucose levels.
- For cement manufacturing, our population would be batches of cement, where we measure the tensile strength and the number of days the cement is cured.
- In the software engineering example, our population consists of all executions of a particular command and control software system, and we observe the failure time of the system in seconds.

In most cases, it is impossible or unrealistic to observe the entire population. For example, some populations have members that do not exist yet (e.g., future batches of cement) or the population is too large (e.g., all pregnant women). So researchers measure only a part of the target population, called a ***sample.*** If we are going to make inferences about the population using the information obtained from a sample, then it is important that the sample be representative of the population. This can usually be accomplished by selecting a ***simple random sample***, where all possible samples are equally likely to be selected.

A random sample of size $n$ is said to be ***independent and identically distributed*** (iid) when the random variables $X_1, X_2, \ldots, X_n$ each have a common probability density (mass) function given by $f(x)$. Additionally, when they are both independent and identically distributed (iid), the joint probability density (mass) function is given by

$$f(x_1, \ldots, x_n) = f(x_1) \times \ldots \times f(x_n),$$

which is simply the product of the individual densities (or mass functions) evaluated at each sample point.

There are two types of simple random sampling: sampling with replacement and sampling without replacement. When we sample with replacement, we select an object, observe the characteristic we are interested in, and return the object to the population. In this case, an object can be selected for the sample more than once. When the sampling is done without replacement, objects can be selected at most one time. These concepts will be used in Chapters 6 and 7 where the bootstrap and other resampling methods are discussed.

Alternative sampling methods exist. In some situations, these methods are more practical and offer better random samples than simple random sampling. One such method, called ***stratified random sampling***, divides the population into levels, and then a simple random sample is taken from each level. Usually, the sampling is done in such a way that the number sampled from each level is proportional to the number of objects of that level that are in the population. Other sampling methods include cluster sampling and systematic random sampling. For more information on these and others, see the book by Levy and Lemeshow [1999].

Sometimes the goal of inferential statistics is to use the sample to estimate or make some statements about a population parameter. Recall from Chapter 2 that a ***parameter*** is a descriptive measure for a population or a distribution of random variables. For example, population parameters that might be of interest include the mean ($\mu$), the standard deviation ($\sigma$), quantiles, proportions, correlation coefficients, etc.

A ***statistic*** is a function of the observed random variables obtained in a random sample and does not contain any unknown population parameters. Often the statistic is used for the following purposes:

- as a point estimate for a population parameter,
- to obtain a confidence interval estimate for a parameter, or
- as a test statistic in hypothesis testing.

Before we discuss some of the common methods for deriving statistics, we present some of the statistics that will be encountered in the remainder of the text. In most cases, we assume that we have a random sample, $X_1, \ldots, X_n$, of independent, identically (iid) distributed random variables.

### Sample Mean and Sample Variance

A familiar statistic is the ***sample mean*** given by

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i. \tag{3.1}$$

To calculate this in MATLAB, one can use the function called **mean**. If the argument to this function is a matrix, then it provides a vector of means, each one corresponding to the mean of a column. One can find the mean along any dimension (**dim**) of multi-dimensional arrays using the syntax: **mean(x,dim)**.

Another statistic that we will see again is the ***sample variance***, calculated from

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2 = \frac{1}{n(n-1)}\left(n\sum_{i=1}^{n} X_i^2 - \left(\sum_{i=1}^{n} X_i\right)^2\right). \tag{3.2}$$

The sample standard deviation is given by the square root of the variance (Equation 3.2) and is denoted by $S$. These statistics can be calculated in MATLAB using the functions **std(x)** and **var(x)**, where **x** is an array containing the sample values. As with the function **mean**, these can have matrices or multi-dimensional arrays as input arguments.

### Sample Moments

The sample moments can be used to estimate the population moments described in Chapter 2. The ***r-th sample moment*** about zero is given by

$$M'_r = \frac{1}{n}\sum_{i=1}^{n} X_i^r. \tag{3.3}$$

Note that the sample mean is obtained when $r = 1$. The $r$-th sample moments about the sample mean are statistics that estimate the population central moments and can be found using the following

$$M_r = \frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^r. \tag{3.4}$$

We can use Equation 3.4 to obtain estimates for the coefficient of skewness $\gamma_1$ and the coefficient of kurtosis $\gamma_2$. Recall that these are given by

$$\gamma_1 = \frac{\mu_3}{\mu_2^{3/2}}, \tag{3.5}$$

and

$$\gamma_2 = \frac{\mu_4}{\mu_2^2}. \tag{3.6}$$

Substituting the sample moments for the population moments in Equations 3.5 and 3.6, we have

$$\hat{\gamma}_1 = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^3}{\left(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2\right)^{3/2}}, \tag{3.7}$$

and

$$\hat{\gamma}_2 = \frac{\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^4}{\left(\frac{1}{n}\sum_{i=1}^{n}(X_i - \bar{X})^2\right)^2}. \tag{3.8}$$

We are using the 'hat' notation to denote an estimate. Thus, $\hat{\gamma}_1$ is an estimate for $\gamma_1$. The following example shows how to use MATLAB to obtain the sample coefficient of skewness and sample coefficient of kurtosis.

### Example 3.1

In this example, we will generate a random sample that is uniformly distributed over the interval (0, 1). We would expect this sample to have a coefficient of skewness close to zero because it is a symmetric distribution. We would expect the kurtosis to be different from 3, because the random sample is not generated from a normal distribution.

```
% Generate a random sample from the uniform
% distribution.
n = 200;
x = rand(1,200);
% Find the mean of the sample.
```

```
mu = mean(x);
% Find the numerator and denominator for gamma_1.
num = (1/n)*sum((x-mu).^3);
den = (1/n)*sum((x-mu).^2);
gam1 = num/den^(3/2);
```

This results in a coefficient of skewness of `gam1 = -0.0542`, which is not too far from zero. Now we find the kurtosis using the following MATLAB commands:

```
% Find the kurtosis.
num = (1/n)*sum((x-mu).^4);
den = (1/n)*sum((x-mu).^2);
gam2 = num/den^2;
```

This gives a kurtosis of `gam2 = 1.8766`, which is not close to 3, as expected. ❑

We note that these statistics might not be the best to use in terms of bias (see Section 3.4). However, they will prove to be useful as examples in Chapters 6 and 7, where we look at bootstrap methods for estimating the bias in a statistic. The MATLAB Statistics Toolbox function called **skewness** returns the coefficient of skewness for a random sample. The function **kurtosis** calculates the sample coefficient of kurtosis (*not* the coefficient of excess kurtosis).

### Covariance

In the definitions given below (Equations 3.9 and 3.10), we assume that all expectations exist. The ***covariance*** of two random variables $X$ and $Y$, with joint probability density function $f(x, y)$, is defined as

$$Cov(X, Y) = \sigma_{X,Y} = E[(X - \mu_X)(Y - \mu_Y)]. \tag{3.9}$$

The ***correlation coefficient*** of $X$ and $Y$ is given by

$$Corr(X, Y) = \rho_{X,Y} = \frac{Cov(X, Y)}{\sigma_X \sigma_Y} = \frac{\sigma_{X,Y}}{\sigma_X \sigma_Y}, \tag{3.10}$$

where $\sigma_X > 0$ and $\sigma_Y > 0$.

The correlation is a measure of the linear relationship between two random variables. If the joint distribution of two variables has a correlation coefficient, then $-1 \leq \rho_{X,Y} \leq 1$. When $\rho_{X,Y} = 1$, then $X$ and $Y$ are perfectly positively correlated. This means that the possible values for $X$ and $Y$ lie on a line with positive slope. On the other hand, when $\rho_{X,Y} = -1$, then the situation is the opposite: $X$ and $Y$ are perfectly negatively correlated. If $X$ and $Y$ are

independent, then $\rho_{X,Y} = 0$. Note that the converse of this statement does not necessarily hold.

There are statistics that can be used to estimate these quantities. Let's say we have a random sample of size $n$ denoted as $(X_1, Y_1), \ldots, (X_n, Y_n)$. The sample covariance is typically calculated using the following statistic

$$\hat{\sigma}_{X,Y} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y}) \ . \tag{3.11}$$

This is the definition used in the MATLAB function **cov**. In some instances, the empirical covariance is used [Efron and Tibshirani, 1993]. This is similar to Equation 3.11, except that we divide by $n$ instead of $n-1$. The sample correlation coefficient for two variables is given by

$$\hat{\rho}_{X,Y} = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{\left(\sum_{i=1}^{n}(X_i - \bar{X})^2\right)^{1/2}\left(\sum_{i=1}^{n}(Y_i - \bar{Y})^2\right)^{1/2}} \ . \tag{3.12}$$

In the next example, we investigate the commands available in MATLAB that return the statistics given in Equations 3.11 and 3.12. It should be noted that the quantity in Equation 3.12 is also bounded below by $-1$ and above by 1.

### Example 3.2

In this example, we show how to use the MATLAB **cov** function to find the covariance between two variables and the **corrcoef** function to find the correlation coefficient. Both of these functions are available in the standard MATLAB language. We use the **cement** data [Hand, et al., 1994], which were analyzed by Hald [1952], to illustrate the basic syntax of these functions. The relationship between the two variables is nonlinear, so Hald looked at the log of the tensile strength as a function of the reciprocal of the drying time. When the **cement** data are loaded, we get a vector **x** representing the drying times and a vector **y** that contains the tensile strength. A scatterplot of the transformed data is shown in Figure 3.1.

```
% First load the data.
load cement
% Now get the transformations.
xr = 1./x;
logy = log(y);
% Now get a scatterplot of the data to see if
% the relationship is linear.
```

```
plot(xr,logy,'x')
axis([0 1.1 2.4 4])
xlabel('Reciprocal of Drying Time')
ylabel('Log of Tensile Strength')
```

We now show how to get the covariance matrix and the correlation coefficient for these two variables.

```
% Now get the covariance and
% the correlation coefficient.
cmat = cov(xr,logy);
cormat = corrcoef(xr,logy);
```

The results are:

```
cmat =
    0.1020   -0.1169
   -0.1169    0.1393
cormat =
   1.0000   -0.9803
   -0.9803    1.0000
```

Note that the sample correlation coefficient (Equation 3.12) is given by the off-diagonal element of `cormat`, $\hat{\rho} = -0.9803$. We see that the variables are negatively correlated, which is what we expect from Figure 3.1 (the log of the tensile strength decreases with increasing reciprocal of drying time).
□

## 3.3 Sampling Distributions

It was stated in the previous section that we sometimes use a statistic calculated from a random sample as a point estimate of a population parameter. For example, we might use $\bar{X}$ to estimate $\mu$ or use $S$ to estimate $\sigma$. Since we are using a sample and not observing the entire population, there will be some error in our estimate. In other words, it is unlikely that the statistic will equal the parameter. To manage the uncertainty and error in our estimate, we must know the sampling distribution for the statistic. The ***sampling distribution*** is the underlying probability distribution for a statistic. To understand the remainder of the text, it is important to remember that *a statistic is a random variable.*

The sampling distributions for many common statistics are known. For example, if our random variable is from the normal distribution, then we know how the sample mean is distributed. Once we know the sampling distribution of our statistic, we can perform statistical hypothesis tests and calculate confidence intervals. If we do not know the distribution of our statistic,

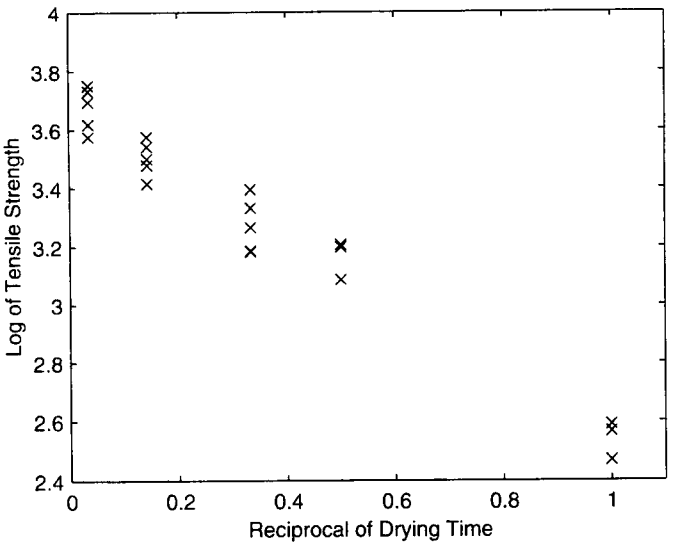



FIGURE 3.1
This scatterplot shows the observed drying times and corresponding tensile strength of the cement. Since the relationship is nonlinear, the variables are transformed as shown here. A linear relationship seems to be a reasonable model for these data.

then we must use Monte Carlo simulation techniques or bootstrap methods to estimate the sampling distribution (see Chapter 6).

To illustrate the concept of a sampling distribution, we discuss the sampling distribution for $\bar{X}$, where the random variable $X$ follows a distribution given by the probability density function $f(x)$. It turns out that the distribution for the sample mean can be found using the Central Limit Theorem.

*CENTRAL LIMIT THEOREM*

*Let $f(x)$ represent a probability density with finite variance $\sigma^2$ and mean $\mu$. Also, let $\bar{X}$ be the sample mean for a random sample of size $n$ drawn from this distribution. For large $n$, the distribution of $\bar{X}$ is approximately normally distributed with mean $\mu$ and variance given by $\sigma^2/n$.*

❑

The Central Limit Theorem states that as the sample size gets large, the distribution of the sample mean approaches the normal distribution regardless of how the random variable $X$ is distributed. However, if we are sampling from a normal population, then the distribution of the sample mean is exactly normally distributed with mean $\mu$ and variance $\sigma^2/n$.

This information is important, because we can use it to determine how much error there is in using $\bar{X}$ as an estimate of the population mean $\mu$. We can also perform statistical hypothesis tests using $\bar{X}$ as a test statistic and can calculate confidence intervals for $\mu$. In this book, we are mainly concerned with computational (rather than theoretical) methods for finding sampling distributions of statistics (e.g., Monte Carlo simulation or resampling). The sampling distribution of $\bar{X}$ is used to illustrate the concepts covered in remaining chapters.

## 3.4 Parameter Estimation

One of the first tasks a statistician or an engineer undertakes when faced with data is to try to summarize or describe the data in some manner. Some of the statistics (sample mean, sample variance, coefficient of skewness, etc.) we covered in Section 3.2 can be used as descriptive measures for our sample. In this section, we look at methods to derive and to evaluate estimates of population parameters.

There are several methods available for obtaining parameter estimates. These include the method of moments, maximum likelihood estimation, Bayes estimators, minimax estimation, Pitman estimators, interval estimates, robust estimation, and many others. In this book, we discuss the maximum likelihood method and the method of moments for deriving estimates for population parameters. These somewhat classical techniques are included as illustrative examples only and are not meant to reflect the state of the art in this area. Many useful (and computationally intensive!) methods are not covered here, but references are provided in Section 3.7. However, we do present some alternative methods for calculating interval estimates using Monte Carlo simulation and resampling methods (see Chapters 6 and 7).

Recall that a sample is drawn from a population that is distributed according to some function whose characteristics are governed by certain parameters. For example, our sample might come from a population that is normally distributed with parameters $\mu$ and $\sigma^2$. Or, it might be from a population that is exponentially distributed with parameter $\lambda$. The goal is to use the sample to estimate the corresponding population parameters. If the sample is representative of the population, then a function of the sample should provide a useful estimate of the parameters.

Before we undertake our discussion of maximum likelihood, we need to define what an estimator is. Typically, population parameters can take on values from a subset of the real line. For example, the population mean can be any real number, $-\infty < \mu < \infty$, and the population standard deviation can be any positive real number, $\sigma > 0$. The set of all possible values for a parameter $\theta$ is called the ***parameter space***. The ***data space*** is defined as the set of all possible values of the random sample of size $n$. The estimate is calculated from

the sample data as a function of the random sample. An ***estimator*** is a function or mapping from the data space to the parameter space and is denoted as

$$T = t(X_1, \ldots, X_n). \tag{3.13}$$

Since an estimator is calculated using the sample alone, it is a statistic. Furthermore, if we have a random sample, then an estimator is also a random variable. This means that the value of the estimator varies from one sample to another based on its sampling distribution. In order to assess the usefulness of our estimator, we need to have some criteria to measure the performance. We discuss four criteria used to assess estimators: bias, mean squared error, efficiency, and standard error. In this discussion, we only present the definitional aspects of these criteria.

### Bias

The bias in an estimator gives a measure of how much error we have, on average, in our estimate when we use $T$ to estimate our parameter $\theta$. The ***bias*** is defined as

$$\text{bias}(T) = E[T] - \theta. \tag{3.14}$$

If the estimator is unbiased, then the expected value of our estimator equals the true parameter value, so $E[T] = \theta$.

To determine the expected value in Equation 3.14, we must know the distribution of the statistic $T$. In these situations, the bias can be determined analytically. When the distribution of the statistic is not known, then we can use methods such as the jackknife and the bootstrap (see Chapters 6 and 7) to estimate the bias of $T$.

### Mean Squared Error

Let $\theta$ denote the parameter we are estimating and $T$ denote our estimate, then the ***mean squared error*** (MSE) of the estimator is defined as

$$\text{MSE}(T) = E[(T - \theta)^2]. \tag{3.15}$$

Thus, the MSE is the expected value of the squared error. We can write this in more useful quantities such as the bias and variance of $T$. (The reader will see this again in Chapter 8 in the context of probability density estimation.) If we expand the expected value on the right hand side of Equation 3.15, then we have

$$\text{MSE}(T) = E[(T^2 - 2T\theta + \theta^2)] = E[T^2] - 2\theta E[T] + \theta^2. \quad (3.16)$$

By adding and subtracting $(E[T])^2$ to the right hand side of Equation 3.16, we have the following

$$\text{MSE}(T) = E[T^2] - (E[T])^2 + (E[T])^2 - 2\theta E[T] + \theta^2. \quad (3.17)$$

The first two terms of Equation 3.17 are the variance of $T$, and the last three terms equal the squared bias of our estimator. Thus, we can write the mean squared error as

$$\begin{aligned} \text{MSE}(T) &= E[T^2] - (E[T])^2 + (E[T] - \theta)^2 \\ &= V(T) + [\text{bias}(T)]^2. \end{aligned} \quad (3.18)$$

Since the mean squared error is based on the variance and the squared bias, the error will be small when the variance and the bias are both small. When $T$ is unbiased, then the mean squared error is equal to the variance only. The concepts of bias and variance are important for assessing the performance of any estimator.

### Relative Efficiency

Another measure we can use to compare estimators is called efficiency, which is defined using the MSE. For example, suppose we have two estimators $T_1 = t_1(X_1, \ldots, X_n)$ and $T_2 = t_2(X_1, \ldots, X_n)$ for the same parameter. If the MSE of one estimator is less than the other (e.g., $\text{MSE}(T_1) < \text{MSE}(T_2)$), then $T_1$ is said to be more efficient than $T_2$.

The ***relative efficiency*** of $T_1$ to $T_2$ is given by

$$eff(T_1, T_2) = \frac{\text{MSE}(T_2)}{\text{MSE}(T_1)}. \quad (3.19)$$

If this ratio is greater than one, then $T_1$ is a more efficient estimator of the parameter.

### Standard Error

We can get a measure of the precision of our estimator by calculating the standard error. The ***standard error*** of an estimator (or a statistic) is defined as the standard deviation of its sampling distribution:

$$SE(T) = \sqrt{V(T)} = \sigma_T.$$

To illustrate this concept, let's use the sample mean as an example. We know that the variance of the estimator is

$$V(\bar{X}) = \frac{1}{n}\sigma^2 ,$$

for large $n$. So, the standard error is given by

$$SE(\bar{X}) = \sigma_{\bar{X}} = \frac{\sigma}{\sqrt{n}} . \tag{3.20}$$

If the standard deviation $\sigma$ for the underlying population is unknown, then we can substitute an estimate for the parameter. In this case, we call it the estimated standard error:

$$\hat{SE}(\bar{X}) = \hat{\sigma}_{\bar{X}} = \frac{S}{\sqrt{n}} . \tag{3.21}$$

Note that the estimate in Equation 3.21 is also a random variable and has a probability distribution associated with it.

If the bias in an estimator is small, then the variance of the estimator is approximately equal to the MSE, $V(T) \approx \text{MSE}(T)$. Thus, we can also use the square root of the MSE as an estimate of the standard error.

### Maximum Likelihood Estimation

A ***maximum likelihood estimator*** is that value of the parameter (or parameters) that maximizes the likelihood function of the sample. The ***likelihood function*** of a random sample of size $n$ from density (mass) function $f(x;\theta)$ is the joint probability density (mass) function, denoted by

$$L(\theta;x_1, \ldots, x_n) = f(x_1, \ldots, x_n;\theta) . \tag{3.22}$$

Equation 3.22 provides the likelihood that the random variables take on a particular value $x_1, \ldots, x_n$. Note that the likelihood function $L$ is a function of the unknown parameter $\theta$, and that we allow $\theta$ to represent a vector of parameters.

If we have a random sample (independent, identically distributed random variables), then we can write the likelihood function as

$$L(\theta) = L(\theta;x_1, \ldots, x_n) = f(x_1;\theta) \times \ldots \times f(x_n;\theta) , \tag{3.23}$$

which is the product of the individual density functions evaluated at each $x_i$ or sample point.

In most cases, to find the value $\hat{\theta}$ that maximizes the likelihood function, we take the derivative of $L$, set it equal to 0 and solve for $\theta$. Thus, we solve the following likelihood equation

$$\frac{d}{d\theta}L(\theta) = 0. \tag{3.24}$$

It can be shown that the likelihood function, $L(\theta)$, and logarithm of the likelihood function, $\ln L(\theta)$, have their maxima at the same value of $\theta$. It is sometimes easier to find the maximum of $\ln L(\theta)$, especially when working with an exponential function. However, keep in mind that a solution to the above equation does not imply that it is a maximum; it could be a minimum. It is important to ensure this is the case before using the result as a maximum likelihood estimator.

When a distribution has more than one parameter, then the likelihood function is a function of all parameters that pertain to the distribution. In these situations, the maximum likelihood estimates are obtained by taking the partial derivatives of the likelihood function (or $\ln L(\theta)$), setting them all equal to zero, and solving the system of equations. The resulting estimators are called the joint maximum likelihood estimators. We see an example of this below, where we derive the maximum likelihood estimators for $\mu$ and $\sigma^2$ for the normal distribution.

## Example 3.3

In this example, we derive the maximum likelihood estimators for the parameters of the normal distribution. We start off with the likelihood function for a random sample of size $n$ given by

$$L(\theta) = \prod_{i=1}^{n} \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{(x_i-\mu)^2}{2\sigma^2}\right\} = \left(\frac{1}{2\pi\sigma^2}\right)^{n/2} \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right).$$

Since this has the exponential function in it, we will take the logarithm to obtain

$$\ln[L(\theta)] = \ln\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}}\right] + \ln\left[\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2\right)\right].$$

This simplifies to

$$\ln[L(\theta)] = -\frac{n}{2}\ln[2\pi] - \frac{n}{2}\ln[\sigma^2] - \frac{1}{2\sigma^2}\sum_{i=1}^{n}(x_i-\mu)^2, \tag{3.25}$$

with $\sigma > 0$ and $-\infty < \mu < \infty$. The next step is to take the partial derivative of Equation 3.25 with respect to $\mu$ and $\sigma^2$. These derivatives are

$$\frac{\partial}{\partial\mu}\ln L = \frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i-\mu), \tag{3.26}$$

and

$$\frac{\partial}{\partial\sigma^2}\ln L = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{n}(x_i-\mu)^2. \tag{3.27}$$

We then set Equations 3.26 and 3.27 equal to zero and solve for $\mu$ and $\sigma^2$. Solving the first equation for $\mu$, we get the familiar sample mean for the estimator.

$$\frac{1}{\sigma^2}\sum_{i=1}^{n}(x_i-\mu) = 0,$$

$$\sum_{i=1}^{n}x_i = n\mu,$$

$$\hat{\mu} = \bar{x} = \frac{1}{n}\sum_{i=1}^{n}x_i.$$

Substituting $\hat{\mu} = \bar{x}$ into Equation 3.27, setting it equal to zero, and solving for the variance, we get

$$\begin{aligned} -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4}\sum_{i=1}^{n}(x_i-\bar{x})^2 &= 0 \\ \hat{\sigma}^2 &= \frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2. \end{aligned} \tag{3.28}$$

These are the sample moments about the sample mean, and it can be verified that these solutions jointly maximize the likelihood function [Lindgren, 1993].
❑

We know that the $E[\bar{X}] = \mu$ [Mood, Graybill and Boes, 1974], so the sample mean is an unbiased estimator for the population mean. However, that is not the case for the maximum likelihood estimate for the variance. It can be shown [Hogg and Craig, 1978] that

$$E[\hat{\sigma}^2] = \frac{(n-1)\sigma^2}{n},$$

so we know (from Equation 3.14) that the maximum likelihood estimate, $\hat{\sigma}^2$, for the variance is biased. If we want to obtain an unbiased estimator for the variance, we simply multiply our maximum likelihood estimator by $n/(n-1)$. This yields the familiar statistic for the sample variance given by

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

### Method of Moments

In some cases, it is difficult finding the maximum of the likelihood function. For example, the gamma distribution has the unknown parameter $t$ that is used in the gamma function, $\Gamma(t)$. This makes it hard to take derivatives and solve the equations for the unknown parameters. The method of moments is one way to approach this problem.

In general, we write the unknown population parameters in terms of the population moments. We then replace the population moments with the corresponding sample moments. We illustrate these concepts in the next example, where we find estimates for the parameters of the gamma distribution.

### Example 3.4

The gamma distribution has two parameters, $t$ and $\lambda$. Recall that the mean and variance are given by $t/\lambda$ and $t/\lambda^2$, respectively. Writing these in terms of the population moments, we have

$$E[X] = \frac{t}{\lambda}, \tag{3.29}$$

and

$$V(X) = E[X^2]-(E[X])^2 = \frac{t}{\lambda^2}. \tag{3.30}$$

The next step is to solve Equations 3.29 and 3.30 for $t$ and $\lambda$. From Equation 3.29, we have $t = \lambda E[X]$, and substituting this in the second equation yields

$$E[X^2]-(E[X])^2 = \frac{\lambda E[X]}{\lambda^2}. \tag{3.31}$$

Rearranging Equation 3.31 gives the following expression for $\lambda$

$$\lambda = \frac{E[X]}{E[X^2]-(E[X])^2}. \tag{3.32}$$

We can now obtain the parameter $t$ in terms of the population moments (substitute Equation 3.32 for $\lambda$ in Equation 3.29) as

$$t = \frac{(E[X])^2}{E[X^2]-(E[X])^2}. \tag{3.33}$$

To get our estimates, we substitute the sample moments for $E[X]$ and $E[X^2]$ in Equations 3.32 and 3.33. This yields

$$\hat{t} = \frac{\bar{X}^2}{\frac{1}{n}\sum_{i=1}^{n} X_i^2 - \bar{X}^2}, \tag{3.34}$$

and

$$\hat{\lambda} = \frac{\bar{X}}{\frac{1}{n}\sum_{i=1}^{n} X_i^2 - \bar{X}^2}. \tag{3.35}$$

❑

In Table 3.1, we provide some suggested point estimates for several of the distributions covered in Chapter 2. This table also contains the names of functions to calculate the estimators. In Section 3.6, we discuss the MATLAB code available in the Statistics Toolbox for calculating maximum likelihood estimates of distribution parameters. The reader is cautioned that the estimators

discussed in this chapter are not necessarily the best in terms of bias, variance, etc.

**TABLE 3.1**

Suggested Point Estimators for Parameters

| Distribution | Suggested Estimator | MATLAB Function |
|---|---|---|
| Binomial<br>Note: $X$ is the number of successes in $n$ trials | $\hat{p} = \frac{X}{n}$ | `csbinpar` |
| Exponential | $\hat{\lambda} = 1/\bar{X}$ | `csexpar` |
| Gamma | $\hat{t} = \bar{X}^2 / \left(\frac{1}{n}\sum X_i^2 - \bar{X}^2\right)$<br>$\hat{\lambda} = \bar{X} / \left(\frac{1}{n}\sum X_i^2 - \bar{X}^2\right)$ | `csgampar` |
| Normal | $\hat{\mu} = \bar{X}$<br>$\hat{\sigma}^2 = S^2$ | `mean`<br>`var` |
| Multivariate Normal | $\hat{\mu}_j = \frac{1}{n}\sum_{i=1}^{n} X_{ij}$<br>$\hat{\Sigma}_{ij} = \frac{n\sum_{k=1}^{n} X_{ik}X_{jk} - \sum_{k=1}^{n} X_{ik}\sum_{k=1}^{n} X_{jk}}{n(n-1)}$ | `mean`<br>`cov` |
| Poisson | $\hat{\lambda} = \bar{X}$ | `cspoipar` |

## 3.5 Empirical Distribution Function

Recall from Chapter 2 that the cumulative distribution function is given by

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(t)dt \tag{3.36}$$

for a continuous random variable and by

$$F(a) = \sum_{x_i \leq a} f(x_i) \tag{3.37}$$

for a discrete random variable. In this section, we examine the sample analog of the cumulative distribution function called the ***empirical distribution function***. When it is not suitable to assume a distribution for the random variable, then we can use the empirical distribution function as an estimate of the underlying distribution. One can call this a ***nonparametric*** estimate of the distribution function, because we are not assuming a specific parametric form for the distribution that generates the random phenomena. In a ***parametric*** setting, we would assume a particular distribution generated the sample and estimate the cumulative distribution function by estimating the appropriate parameters.

The empirical distribution function is based on the ***order statistics***. The order statistics for a sample are obtained by putting the data in ascending order. Thus, for a random sample of size $n$, the order statistics are defined as

$$X_{(1)} \leq X_{(2)} \leq \ldots \leq X_{(n)},$$

with $X_{(i)}$ denoting the $i$-th order statistic. The order statistics for a random sample can be calculated easily in MATLAB using the `sort` function.

The empirical distribution function $\hat{F}_n(x)$ is defined as the number of data points less than or equal to $x$ ($\#(X_i \leq x)$) divided by the sample size $n$. It can be expressed in terms of the order statistics as follows

$$\hat{F}_n(x) = \begin{cases} 0; & x < X_{(1)} \\ j/n; & X_{(j)} \leq x < X_{(j+1)} \\ 1; & x \geq X_{(n)}. \end{cases} \tag{3.38}$$

Figure 3.2 illustrates these concepts. We show the empirical cumulative distribution function for a standard normal and include the theoretical distribution function to verify the results. In the following section, we describe a descriptive measure for a population called a quantile, along with its corresponding estimate. Quantiles are introduced here, because they are based on the cumulative distribution function.

## Quantiles

Quantiles have a fundamental role in statistics. For example, they can be used as a measure of central tendency and dispersion, they provide the critical val-

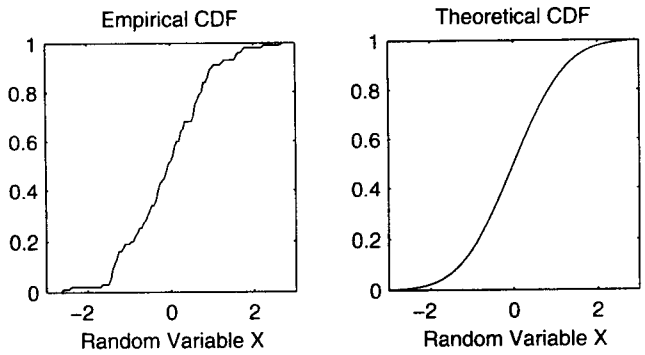



FIGURE 3.2
This shows the theoretical and empirical distribution functions for a standard normal distribution.

ues in hypothesis testing (see Chapter 6), and they are used in exploratory data analysis for assessing distributions (see Chapter 5).

The ***quantile*** $q_p$ of a random variable (or equivalently of its distribution) is defined as the smallest number $q$ such that the cumulative distribution function is greater than or equal to some $p$, where $0 < p < 1$. This can be calculated for a continuous random variable with density function $f(x)$ by solving

$$p = \int_{-\infty}^{q_p} f(x)dx \tag{3.39}$$

for $q_p$, or by using the inverse of the cumulative distribution function,

$$q_p = F^{-1}(p). \tag{3.40}$$

Stating this another way, the $p$-th quantile of a random variable $X$ is the value $q_p$ such that

$$F(q_p) = P(X \leq q_p) = p \tag{3.41}$$

for $0 < p < 1$.

Some well known examples of quantiles are the ***quartiles***. These are denoted by $q_{0.25}$, $q_{0.5}$, and $q_{0.75}$. In essence, these divide the distribution into four equal (in terms of probability or area under the curve) segments. The second quartile is also called the ***median*** and satisfies

$$0.5 = \int_{-\infty}^{q_{0.5}} f(x)dx . \tag{3.42}$$

We can get a measure of the dispersion of the random variable by looking at the ***interquartile range*** (IQR) given by

$$\text{IQR} = q_{0.75} - q_{0.25} . \tag{3.43}$$

One way to obtain an estimate of the quantiles is based on the empirical distribution function. If we let $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$ denote the order statistics for a random sample of size $n$, then $X_{(j)}$ is an estimate of the $(j - 0.5)/n$ quantile [Banks, 2001; Cleveland, 1993]:

$$X_{(j)} \approx F^{-1}\left(\frac{j-0.5}{n}\right). \tag{3.44}$$

We are not limited to a value of 0.5 in Equation 3.44. In general, we can estimate the $p$-th quantile using the following

$$\hat{q}_p = X_{(j)}; \qquad \frac{j-1}{n} < p \leq \frac{j}{n}; \qquad j = 1, \ldots, n . \tag{3.45}$$

As already stated, Equation 3.45 is not the only way to estimate quantiles. For more information on other methods, see Kotz and Johnson [Vol. 7, 1986]. The analyst should exercise caution when calculating quartiles (or other quantiles) using computer packages. Statistical software packages define them differently [Frigge, Hoaglin, and Iglewicz, 1989], so these statistics might vary depending on the formulas that are used.

### EXAMPLE 3.5

In this example, we will show one way to determine the sample quartiles. The second sample quartile $\hat{q}_{0.5}$ is the sample median of the data set. We can calculate this using the function **median**. We could calculate the first quartile $\hat{q}_{0.25}$ as the median of the ordered data that are at the median or below. The third quartile $\hat{q}_{0.75}$ would be calculated as the median of the data that are at $\hat{q}_{0.5}$ or above. The following MATLAB code illustrates these concepts.

```
% Generate the random sample and sort.
x = sort(rand(1,100));
% Find the median of the lower half - first quartile.
q1 = median(x(1:50));
% Find the median.
q2 = median(x);
```

```
% Find the median of the upper half - third quartile.
q3 = median(x(51:100));
```

The quartiles obtained from this random sample are:

```
q1 = 0.29, q2 = 0.53, q3 = 0.79
```

The theoretical quartiles for the uniform distribution are $q_{0.25} = 0.25$, $q_{0.5} = 0.5$, and $q_{0.75} = 0.75$. So we see that the estimates seem reasonable.
❑

Equation 3.44 provides one way to estimate the quantiles from a random sample. In some situations, we might need to determine an estimate of a quantile that does not correspond to $(j - 0.5)/n$. For instance, this is the case when we are constructing q-q plots (see Chapter 5), and the sample sizes differ. We can use interpolation to find estimates of quantiles that are not represented by Equation 3.44.

### Example 3.5

The MATLAB function **interp1** (in the standard package) returns the interpolated value $Y_I$ at a given $X_I$, based on some observed values $X_{obs}$ and $Y_{obs}$. The general syntax is

```
yint = interp1(xobs, yobs, xint);
```

In our case, the argument of $F^{-1}$ in Equation 3.44 represents the observed values $X_{obs}$, and the order statistics $X_{(j)}$ correspond to the $Y_{obs}$. The MATLAB code for this procedure is shown below.

```
% First generate some standard normal data.
x = randn(500,1);
% Now get the order statistics. These will serve
% as the observed values for the ordinate (Y_obs).
xs = sort(x);
% Now get the observed values for the abscissa (X_obs).
n=length(x);
phat = ((1:n)-0.5)/n;
% We want to get the quartiles.
p = [0.25, 0.5, 0.75];
% The following provides the estimates of the quartiles
% using linear interpolation.
qhat = interp1(phat,xs,p);
```

The resulting estimates are

```
qhat = -0.6928    0.0574    0.6453.
```

The reader is asked to explore this further in the exercises.
❑

## 3.6 MATLAB Code

The MATLAB Statistics Toolbox has functions for calculating the maximum likelihood estimates for most of the common distributions, including the gamma and the Weibull distributions. It is important to remember that the parameters estimated for some of the distributions (e.g., exponential and gamma) are different from those defined in Chapters 2 and 3. We refer the reader to Appendix E for a complete list of the functions appropriate to this chapter. Table 3.2 provides a partial list of MATLAB functions for calculating statistics.We also provide some functions for statistics with the Computational Statistics Toolbox. These are summarized in Table 3.3.

TABLE 3.2

List of MATLAB functions for calculating statistics

| Purpose | MATLAB Function |
|---|---|
| These functions are available in the standard MATLAB package. | `mean`<br>`var`<br>`std`<br>`cov`<br>`median`<br>`corrcoef`<br>`max, min`<br>`sort` |
| These functions for calculating descriptive statistics are available in the MATLAB Statistics Toolbox. | `harmmean`<br>`iqr`<br>`kurtosis`<br>`mad`<br>`moment`<br>`prctile`<br>`range`<br>`skewness`<br>`trimmean` |
| These MATLAB Statistics Toolbox functions provide the maximum likelihood estimates for distributions. | `betafit`<br>`binofit`<br>`expfit`<br>`gamfit`<br>`normfit`<br>`poissfit`<br>`weibfit`<br>`unifit`<br>`mle` |

TABLE 3.3

List of Functions from Chapter 3 Included in the Computational Statistics Toolbox

| Purpose | MATLAB Function |
|---|---|
| These functions are used to obtain parameter estimates for a distribution. | `csbinpar`<br>`csexpar`<br>`csgampar`<br>`cspoipar`<br>`csunipar` |
| These functions return the quantiles. | `csbinoq`<br>`csexpoq`<br>`csunifq`<br>`csweibq`<br>`csnormq`<br>`csquantiles` |
| Other descriptive statistics | `csmomentc`<br>`cskewness`<br>`cskurtosis`<br>`csmoment`<br>`csecdf` |

## 3.7 Further Reading

Many books discuss sampling distributions and parameter estimation. These topics are covered at an undergraduate level in most introductory statistics books for engineers or non-statisticians. For the advanced undergraduate and beginning graduate student, we recommend the text on mathematical statistics by Hogg and Craig [1978]. Another excellent introductory book on mathematical statistics that contains many applications and examples is written by Mood, Graybill and Boes [1974]. Other texts at this same level include Bain and Engelhardt [1992], Bickel and Doksum [2001], and Lindgren [1993]. For the reader interested in the theory of point estimation on a more advanced graduate level, the book by Lehmann and Casella [1998] and Lehmann [1994] are classics.

Most of the texts already mentioned include descriptions of other methods (Bayes methods, minimax methods, Pitman estimators, etc.) for estimating parameters. For an introduction to robust estimation methods, see the books by Wilcox [1997], Launer and Wilkinson [1979], Huber [1981], or Rousseeuw and Leroy [1987] or see the survey paper by Hogg [1974]. Finally, the text by

Keating, Mason and Sen [1993] provides an introduction to Pitman's measure of closeness as a way to assess the performance of competing estimators.

## Exercises

3.1. Generate 500 random samples from the standard normal distribution for sample sizes of $n$ = 2, 15, and 45. At each sample size, calculate the sample mean for all 500 samples. How are the means distributed as $n$ gets large? Look at a histogram of the sample means to help answer this question. What is the mean and variance of the sample means for each $n$? Is this what you would expect from the Central Limit Theorem? Here is some MATLAB code to get you started.

For each $n$:

```
% Generate 500 random samples of size n:
x = randn(n, 500);
% Get the mean of each sample:
xbar = mean(x);
% Do a histogram with superimposed normal density.
% This function is in the MATLAB Statistics Toolbox.
% If you do not have this, then just use the
% function hist instead of histfit.
histfit(xbar);
```

3.2. Repeat problem 3.1 for random samples drawn from a uniform distribution. Use the MATLAB function **rand** to get the samples.

3.3. We have two unbiased estimators $T_1$ and $T_2$ of the parameter $\theta$. The variances of the estimators are given by $V(T_2) = 8$ and $V(T_1) = 4$. What is the MSE of the estimators? Which estimator is better and why? What is the relative efficiency of the two estimators?

3.4. Repeat Example 3.1 using different sample sizes. What happens to the coefficient of skewness and kurtosis as the sample size gets large?

3.5. Repeat Example 3.1 using samples generated from a standard normal distribution. You can use the MATLAB function **randn** to generate your samples. What happens to the coefficient of skewness and kurtosis as the sample size gets large?

3.6. Generate a random sample that is uniformly distributed over the interval (0, 1). Plot the empirical distribution function over the interval (-0.5, 1.5). There is also a function in the Statistics Toolbox called **cdfplot** that will do this.

3.7. Generate a random sample of size 100 from a normal distribution with mean 10 and variance of 2 (use `randn(1,100)*sqrt(2)+10`). Plot the empirical cumulative distribution function. What is the value of the empirical distribution function evaluated at a point less than

the smallest observation in your random sample? What is the value of the empirical cumulative distribution function evaluated at a point that is greater than the largest observation in your random sample?

3.8. Generate a random sample of size 100 from a normal distribution. What are the estimated quartiles?

3.9. Generate a random sample of size 100 from a uniform distribution (use the MATLAB function **rand** to generate the samples). What are the sample quantiles for $p = 0.33, 0.40, 0.63, 0.90$? Is this what you would expect from theory?

3.10. Write a MATLAB function that will return the sample quartiles based on the general definition given for sample quantiles (Equation 3.44).

3.11. Repeat Examples 3.5 and 3.6 for larger sample sizes. Do your estimates for the quartiles get closer to the theoretical values?

3.12. Derive the median for an exponential random variable.

3.13. Calculate the quartiles for the exponential distribution.

3.14. Compare the values obtained for the estimated quartiles in Example 3.6 with the theoretical quantities. You can find the theoretical quantities using **norminv**. Increase the sample size to $n = 1000$. Does your estimate get better?

3.15. Another measure of skewness, called the ***quartile coefficient of skewness***, for a sample is given by

$$\hat{\gamma}_{1_q} = \frac{\hat{q}_{0.75} - 2\hat{q}_{0.5} + \hat{q}_{0.25}}{\hat{q}_{0.75} - \hat{q}_{0.25}}.$$

Write a MATLAB function that returns this statistic.

3.16. Investigate the bias in the maximum likelihood estimate of the variance that is given in Equation 3.28. Generate a random sample from the standard normal distribution. You can use the **randn** function that is available in the standard MATLAB package. Calculate $\hat{\sigma}^2$ using Equation 3.28 and record the value in a vector. Repeat this process (generate a random sample from the standard normal distribution, estimate the variance, save the value) many times. Once you are done with this procedure, you should have many estimates for the variance. Take the mean of these estimates to get an estimate of the expected value of $\hat{\sigma}^2$. How does this compare with the known value of $\sigma^2 = 1$? Does this indicate that the maximum likelihood estimate for the variance is biased? What is the estimated bias from this procedure?

# Chapter 4

## *Generating Random Variables*

### 4.1 Introduction

Many of the methods in computational statistics require the ability to generate random variables from known probability distributions. This is at the heart of Monte Carlo simulation for statistical inference (Chapter 6), bootstrap and resampling methods (Chapters 6 and 7), Markov chain Monte Carlo techniques (Chapter 11), and the analysis of spatial point processes (Chapter 12). In addition, we use simulated random variables to explain many other topics in this book, such as exploratory data analysis (Chapter 5), density estimation (Chapter 8), and statistical pattern recognition (Chapter 9).

There are many excellent books available that discuss techniques for generating random variables and the underlying theory; references will be provided in the last section. Our purpose in covering this topic is to give the reader the tools they need to generate the types of random variables that often arise in practice and to provide examples illustrating the methods. We first discuss general techniques for generating random variables, such as the inverse transformation and acceptance-rejection methods. We then provide algorithms and MATLAB code for generating random variables for some useful distributions.

### 4.2 General Techniques for Generating Random Variables

#### Uniform Random Numbers

Most methods for generating random variables start with random numbers that are uniformly distributed on the interval $(0, 1)$. We will denote these random variables by the letter $U$. With the advent of computers, we now have

the ability to generate uniform random variables very easily. However, we have to caution the reader that the numbers generated by computers are really pseudorandom because they are generated using a deterministic algorithm. The techniques used to generate uniform random variables have been widely studied in the literature, and it has been shown that some generators have serious flaws [Gentle, 1998].

The basic MATLAB program has a function **rand** for generating uniform random variables. There are several optional arguments, and we take a moment to discuss them because they will be useful in simulation. The function **rand** with no arguments returns a single instance of the random variable $U$. To get an $m \times n$ array of uniform variates, you can use the syntax **rand(m,n)**. A note of caution: if you use **rand(n)**, then you get an $n \times n$ matrix.

The sequence of random numbers that is generated in MATLAB depends on the seed or the state of the generator. The state is reset to the default when it starts up, so the same sequences of random variables are generated whenever you start MATLAB. This can sometimes be an advantage in situations where we would like to obtain a specific random sample, as we illustrate in the next example. If you call the function using **rand('state',0)**, then MATLAB resets the generator to the initial state. If you want to specify another state, then use the syntax **rand('state',j)** to set the generator to the $j$-th state. You can obtain the current state using **S = rand('state')**, where **S** is a 35 element vector. To reset the state to this one, use **rand('state',S)**.

It should be noted that random numbers that are uniformly distributed over an interval $a$ to $b$ may be generated by a simple transformation, as follows

$$X = (b-a) \cdot U + a. \tag{4.1}$$

### Example 4.1

In this example, we illustrate the use of MATLAB's function **rand**.

```
% Obtain a vector of uniform random variables in (0,1).
x = rand(1,1000);
% Do a histogram to plot.
% First get the height of the bars.
[N,X] = hist(x,15);
% Use the bar function to plot.
bar(X,N,1,'w')
title('Histogram of Uniform Random Variables')
xlabel('X')
ylabel('Frequency')
```

The resulting histogram is shown in Figure 4.1. In some situations, the analyst might need to reproduce results from a simulation, say to verify a con-

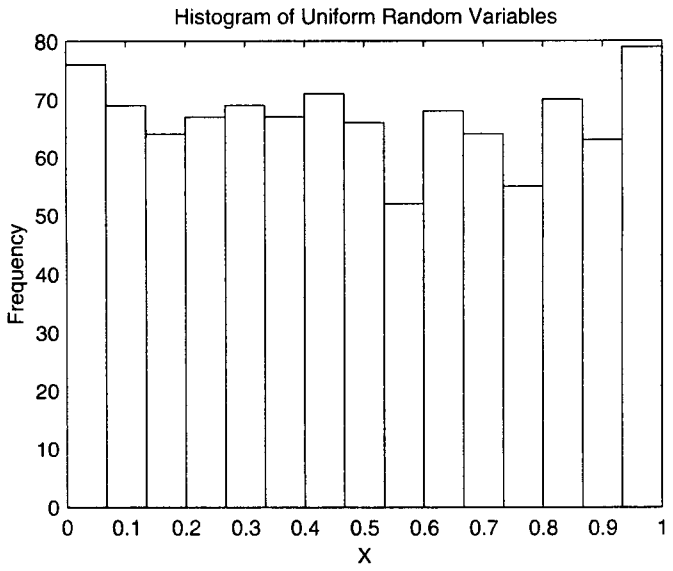



**FIGURE 4.1**
This figure shows a histogram of a random sample from the uniform distribution on the interval (0, 1).

clusion or to illustrate an interesting sample. To accomplish this, the state of the uniform random number generator should be specified at each iteration of the loop. This is accomplished in MATLAB as shown below.

```
% Generate 3 random samples of size 5.
x = zeros(3,5);   % Allocate the memory.
for i = 1:3
   rand('state',i) % set the state
   x(i,:) = rand(1,5);
end
```

The three sets of random variables are

```
0.9528    0.7041    0.9539    0.5982    0.8407
0.8752    0.3179    0.2732    0.6765    0.0712
0.5162    0.2252    0.1837    0.2163    0.4272
```

We can easily recover the five random variables generated in the second sample by setting the state of the random number generator, as follows

```
rand('state',2)
xt = rand(1,5);
```

From this, we get

```
xt = 0.8752    0.3179    0.2732    0.6765    0.0712
```

which is the same as before.
❑

### Inverse Transform Method

The inverse transform method can be used to generate random variables from a continuous distribution. It uses the fact that the cumulative distribution function $F$ is uniform $(0, 1)$ [Ross, 1997]:

$$U = F(X). \tag{4.2}$$

If $U$ is a uniform $(0, 1)$ random variable, then we can obtain the desired random variable $X$ from the following relationship

$$X = F^{-1}(U). \tag{4.3}$$

We see an example of how to use the inverse transform method when we discuss generating random variables from the exponential distribution (see Example 4.6). The general procedure for the inverse transformation method is outlined here.

*PROCEDURE - INVERSE TRANSFORM METHOD (CONTINUOUS)*

1. Derive the expression for the inverse distribution function $F^{-1}(U)$.
2. Generate a uniform random number $U$.
3. Obtain the desired $X$ from $X = F^{-1}(U)$.

This same technique can be adapted to the discrete case [Banks, 2001]. Say we would like to generate a discrete random variable $X$ that has a probability mass function given by

$$P(X = x_i) = p_i; \qquad x_0 < x_1 < x_2 < \ldots; \qquad \sum_i p_i = 1. \tag{4.4}$$

We get the random variables by generating a random number $U$ and then deliver the random number $X$ according to the following

$$X = x_i, \qquad \text{if} \quad F(x_{i-1}) < U \leq F(x_i). \tag{4.5}$$

We illustrate this procedure using a simple example.

### Example 4.2

We would like to simulate a discrete random variable $X$ that has probability mass function given by

$$\begin{aligned} P(X = 0) &= 0.3, \\ P(X = 1) &= 0.2, \\ P(X = 2) &= 0.5. \end{aligned}$$

The cumulative distribution function is

$$F(x) = \begin{cases} 0; & x < 0 \\ 0.3; & 0 \le x < 1 \\ 0.5; & 1 \le x < 2 \\ 1.0; & 2 \le x. \end{cases}$$

We generate random variables for $X$ according to the following scheme

$$X = \begin{cases} 0; & U \le 0.3 \\ 1; & 0.3 < U \le 0.5 \\ 2; & 0.5 < U \le 1. \end{cases}$$

This is easily implemented in MATLAB and is left as an exercise. The procedure is illustrated in Figure 4.2, for the situation where a uniform random variable 0.73 was generated. Note that this would return the variate $x = 2$. ❑

We now outline the algorithmic technique for this procedure. This will be useful when we describe a method for generating Poisson random variables.

*PROCEDURE - INVERSE TRANSFORM (DISCRETE)*

1. Define a probability mass function for $x_i$, $i = 1, \ldots, k$. Note that $k$ could grow infinitely.
2. Generate a uniform random number $U$.
3. If $U \le p_0$ deliver $X = x_0$
4. else if $U \le p_0 + p_1$ deliver $X = x_1$
5. else if $U \le p_0 + p_1 + p_2$ deliver $X = x_2$

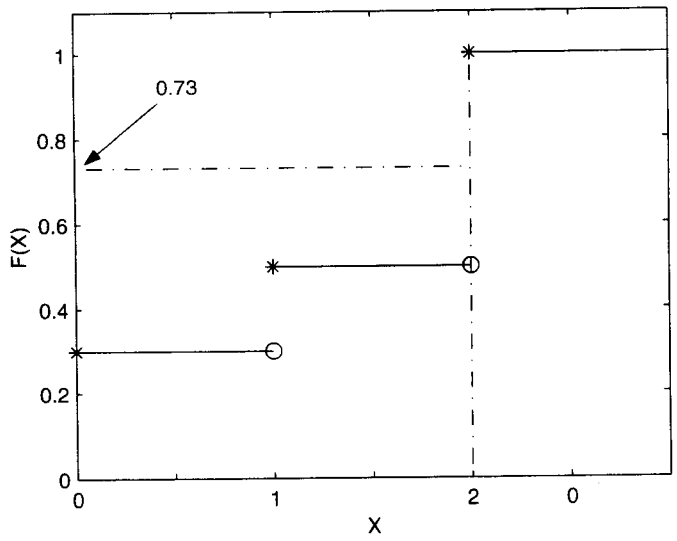



FIGURE 4.2
This figure illustrates the inverse transform procedure for generating discrete random variables. If we generate a uniform random number of $u = 0.73$, then this yields a random variable of $x = 2$.

6. ... else if $U \leq p_0 + \ldots + p_k$ deliver $X = x_k$.

## Example 4.3

We repeat the previous example using this new procedure and implement it in MATLAB. We first generate 100 variates from the desired probability mass function.

```
% Set up storage space for the variables.
X = zeros(1,100);
% These are the x's in the domain.
x = 0:2;
% These are the probability masses.
pr = [0.3 0.2 0.5];
% Generate 100 rv's from the desired distribution.
for i = 1:100
   u = rand;  % Generate the U.
   if u <= pr(1)
      X(i) = x(1);
   elseif u <= sum(pr(1:2))
     % It has to be between 0.3 and 0.5.
      X(i) = x(2);
```

```
    else
       X(i) = x(3); % It has to be between 0.5 and 1.
    end
end
```

One way to verify that our random variables are from the desired distribution is to look at the relative frequency of each $x$.

```
% Find the proportion of each number.
x0 = length(find(X==0))/100;
x1 = length(find(X==1))/100;
x2 = length(find(X==2))/100;
```

The resulting estimated probabilities are

$$\hat{P}(x = x_0) = 0.26$$
$$\hat{P}(x = x_1) = 0.21$$
$$\hat{P}(x = x_2) = 0.53.$$

These values are reasonable when compared with the desired probability mass values.
❑

### Acceptance-Rejection Method

In some cases, we might have a simple method for generating a random variable from one density, say $g(y)$, instead of the density we are seeking. We can use this density to generate from the desired continuous density $f(x)$. We first generate a random number $Y$ from $g(y)$ and accept the value with a probability proportional to the ratio $f(Y)/(g(Y))$.

If we define $c$ as a constant that satisfies

$$\frac{f(y)}{g(y)} \leq c; \qquad \text{for all } y, \tag{4.6}$$

then we can generate the desired variates using the procedure outlined below. The constant $c$ is needed because we might have to adjust the height of $g(y)$ to ensure that it is above $f(y)$. We generate points from $cg(y)$, and those points that are inside the curve $f(y)$ are accepted as belonging to the desired density. Those that are outside are rejected. It is best to keep the number of rejected variates small for maximum efficiency.

*PROCEDURE - ACCEPTANCE-REJECTION METHOD (CONTINUOUS)*

1. Choose a density $g(y)$ that is easy to sample from.
2. Find a constant $c$ such that Equation 4.6 is satisfied.
3. Generate a random number $Y$ from the density $g(y)$.
4. Generate a uniform random number $U$.
5. If

$$U \leq \frac{f(Y)}{cg(Y)},$$

then accept $X = Y$, else go to step 3.

### Example 4.4

We shall illustrate the acceptance-rejection method by generating random variables from the beta distribution with parameters $\alpha = 2$ and $\beta = 1$ [Ross, 1997]. This yields the following probability density function

$$f(x) = 2x; \qquad 0 < x < 1. \tag{4.7}$$

Since the domain of this density is 0 to 1, we use the uniform distribution for our $g(y)$. We must find a constant that we can use to inflate the uniform so it is above the desired beta density. This constant is given by the maximum value of the density function, and from Equation 4.7, we see that $c = 2$. For more complicated functions, techniques from calculus or the MATLAB function **fminsearch** may be used. The following MATLAB code generates 100 random variates from the desired distribution. We save both the accepted and the rejected variates for display purposes only.

```
c = 2;   % constant
n = 100;  % Generate 100 random variables.
% Set up the arrays to store variates.
x = zeros(1,n);  % random variates
xy = zeros(1,n);% corresponding y values
rej = zeros(1,n);% rejected variates
rejy = zeros(1,n); % corresponding y values
irv = 1;
irej = 1;
while irv <= n
   y = rand(1);  % random number from g(y)
   u = rand(1);  % random number for comparison
   if u <= 2*y/c;
      x(irv) = y;
      xy(irv) = u*c;
```

```
        irv = irv+1
    else
        rej(irej) = y;
        rejy(irej) = u*c; % really comparing u*c<=2*y
        irej = irej + 1
    end
end
```

In Figure 4.3, we show the accepted and rejected random variates that were generated in this process. Note that the accepted variates are those that are less than $f(x)$.

❑

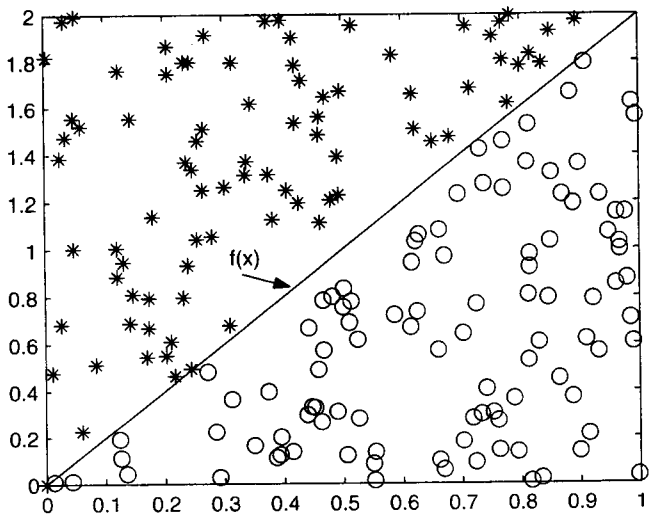



FIGURE 4.3
This shows the points that were accepted ('o') as being generated by $f(x) = 2x$ and those points that were rejected ('*'). The curve represents $f(x)$, so we see that the accepted variates are the ones below the curve.

We can easily adapt this method to generate random variables from a discrete distribution. Here we have a method for simulating a random variable with a probability mass function $q_i = P(Y = i)$, and we would like to obtain a random variable $X$ having a probability mass function $p_i = P(X = i)$. As in the continuous case, we generate a random variable $Y$ from $q_i$ and accept this value with probability $p_Y/(cq_Y)$.

*PROCEDURE - REJECTION METHOD (DISCRETE)*

1. Choose a probability mass function $q_i$ that is easy to sample from.
2. Find a constant $c$ such that $p_Y < cq_Y$.
3. Generate a random number $Y$ from the density $q_i$.
4. Generate a uniform random number $U$.
5. If

$$U \leq \frac{p_Y}{cq_Y},$$

then deliver $X = Y$, else go to step 3.

## Example 4.5

In this example, we use the discrete form of the acceptance-rejection method to generate random variables according to the probability mass function defined as follows

$$\begin{aligned} P(X = 1) &= 0.15, \\ P(X = 2) &= 0.22, \\ P(X = 3) &= 0.33, \\ P(X = 4) &= 0.10, \\ P(X = 5) &= 0.20. \end{aligned}$$

We let $q_Y$ be the discrete uniform distribution on $1, \ldots, 5$, where the probability mass function is given by

$$q_y = \frac{1}{5}; \qquad y = 1, \ldots, 5.$$

We describe a method for generating random variables from the discrete uniform distribution in a later section. The value for $c$ is obtained as the maximum value of $p_y/q_y$, which is 1.65. This quantity is obtained by taking the maximum $p_y$, which is $P(X = 3) = 0.33$, and dividing by $1/5$:

$$\frac{max(p_y)}{1/5} = 0.33 \times 5 = 1.65.$$

The steps for generating the variates are:

1. Generate a variate $Y$ from the discrete uniform density on $1, \ldots, 5$. (One could use the MATLAB Statistics Toolbox function **unidrnd** or **csdunrnd**.)
2. Generate a uniform random number $U$.
3. If

$$U \leq \frac{p_Y}{cq_Y} = \frac{p_Y}{1.65 \cdot 1/5} = \frac{p_Y}{0.33},$$

then deliver $X = Y$, else return to step 1.

The implementation of this example in MATLAB is left as an exercise. ❑

## 4.3 Generating Continuous Random Variables

### Normal Distribution

The main MATLAB program has a function that will generate numbers from the standard normal distribution, so we do not discuss any techniques for generating random variables from the normal distribution. For the reader who is interested in how normal random variates can be generated, most of the references provided in Section 4.6 contain this information.

The MATLAB function for generating standard normal random variables is called **randn**, and its functionality is similar to the function **rand** that was discussed in the previous section. As with the uniform random variable $U$, we can obtain a normal random variable $X$ with mean $\mu$ and variance $\sigma^2$ by means of a transformation. Letting $Z$ represent a standard normal random variable (possibly generated from **randn**), we get the desired $X$ from the relationship

$$X = Z \cdot \sigma + \mu. \tag{4.8}$$

### Exponential Distribution

The inverse transform method can be used to generate random variables from the exponential distribution and serves as an example of this procedure. The distribution function for an exponential random variable with parameter $\lambda$ is given by

$$F(x) = 1 - e^{-\lambda x}; \qquad 0 < x < \infty. \tag{4.9}$$

Letting

$$u = F(x) = 1 - e^{-\lambda x}, \tag{4.10}$$

we can solve for $x$, as follows

$$\begin{aligned} u &= 1 - e^{-\lambda x} \\ e^{-\lambda x} &= 1 - u \\ -\lambda x &= \log(1-u) \\ x &= -\frac{1}{\lambda}\log(1-u). \end{aligned}$$

By making note of the fact that $1 - u$ is also uniformly distributed over the interval (0,1), we can generate exponential random variables with parameter $\lambda$ using the transformation

$$X = -\frac{1}{\lambda}\log(U). \tag{4.11}$$

### Example 4.6

The following MATLAB code will generate exponential random variables for a given $\lambda$.

```
% Set up the parameters.
lam = 2;
n = 1000;
% Generate the random variables.
uni = rand(1,n);
X = -log(uni)/lam;
```

We can generate a set of random variables and plot them to verify that the function does yield exponentially distributed random variables. We plot a histogram of the results along with the theoretical probability density function in Figure 4.4. The MATLAB code given below shows how we did this.

```
% Get the values to draw the theoretical curve.
x = 0:.1:5;
% This is a function in the Statistics Toolbox.
y = exppdf(x,1/2);
% Get the information for the histogram.
[N,h] = hist(X,10);
% Change bar heights to make it correspond to
```

```
% the theoretical density - see Chapter 5.
N = N/(h(2)-h(1))/n;
% Do the plots.
bar(h,N,1,'w')
hold on
plot(x,y)
hold off
xlabel('X')
ylabel('f(x) - Exponential')
```

❑

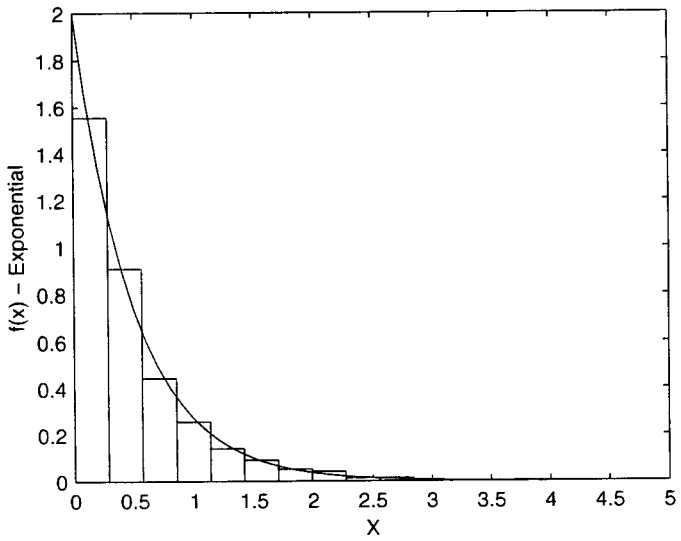



FIGURE 4.4
This shows a probability density histogram of the random variables generated in Example 4.6. We also superimpose the curve corresponding to the theoretical probability density function with $\lambda = 2$. The histogram and the curve match quite well.

## Gamma

In this section, we present an algorithm for generating a gamma random variable with parameters $(t, \lambda)$, where $t$ is an integer. Recall that it has the following distribution function

$$F(x) = \int_0^{\lambda x} \frac{e^{-y} y^{t-1}}{(t-1)!} dy . \tag{4.12}$$

The inverse transform method cannot be used in this case, because a simple closed form solution for its inverse is not possible. It can be shown [Ross, 1997] that the sum of $t$ independent exponentials with the same parameter $\lambda$ is a gamma random variable with parameters $t$ and $\lambda$. This leads to the following transformation based on $t$ uniform random numbers,

$$X = -\frac{1}{\lambda}\log U_1 - \ldots - \frac{1}{\lambda}\log U_t . \tag{4.13}$$

We can simplify this and compute only one logarithm by using a familiar relationship of logarithms. This yields the following

$$X = -\frac{1}{\lambda}\log(U_1 \times \ldots \times U_t) = -\frac{1}{\lambda}\log\left(\prod_{i=1}^{t} U_i\right). \tag{4.14}$$

### Example 4.7

The MATLAB code given below implements the algorithm described above for generating gamma random variables, when the parameter $t$ is an integer.

```
n = 1000;
t = 3;
lam = 2;
% Generate the uniforms needed. Each column
% contains the t uniforms for a realization of a
% gamma random variable.
U = rand(t,n);
% Transform according to Equation 4.13.
% See Example 4.8 for an illustration of Equation 4.14.
logU = -log(U)/lam;
X = sum(logU);
```

To see whether the implementation of the algorithm is correct, we plot them in a probability density histogram.

```
% Now do the histogram.
[N,h] = hist(X,10);
% Change bar heights.
N = N/(h(2)-h(1))/n;
% Now get the theoretical probability density.
% This is a function in the Statistics Toolbox.
x = 0:.1:6;
```

```
y = gampdf(x,t,1/lam);
bar(h,N,1,'w')
hold on
plot(x,y,'k')
hold off
```

The histogram and the corresponding theoretical probability density function are shown in Figure 4.5.
❑

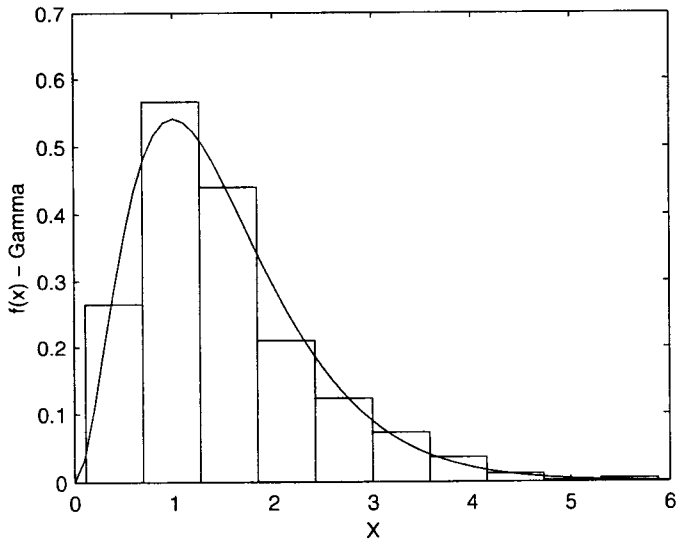



FIGURE 4.5
This shows the probability density histogram for a set of gamma random variables with $t = 3$ and $\lambda = 2$.

### Chi-Square

A chi-square random variable with $\nu$ degrees of freedom is a special case of the gamma distribution, where $\lambda = 1/2$, $t = \nu/2$ and $\nu$ is a positive integer. This can be generated using the gamma distribution method described above with one change. We have to make this change, because the method we presented for generating gamma random variables is for integer $t$, which works for even values of $\nu$.

When $\nu$ is even, say $2k$, we can obtain a chi-square random variable from

$$X = -2\log\left(\prod_{i=1}^{k} U_i\right). \tag{4.15}$$

When $\nu$ is odd, say $2k+1$, we can use the fact that the chi-square distribution with $\nu$ degrees of freedom is the sum of $\nu$ squared independent standard normals [Ross, 1997]. We obtain the required random variable by first simulating a chi-square with $2k$ degrees of freedom and adding a squared standard normal variate $Z$, as follows

$$X = Z^2 - 2\log\left(\prod_{i=1}^{k} U_i\right). \tag{4.16}$$

### Example 4.8

In this example, we provide a function that will generate chi-square random variables.

```
% function X = cschirnd(n,nu)
% This function will return n chi-square
% random variables with degrees of freedom nu.

function X = cschirnd(n,nu)
% Generate the uniforms needed.
rm = rem(nu,2);
k = floor(nu/2);
if rm == 0   % then even degrees of freedom
   U = rand(k,n);
   if k ~= 1
      X = -2*log(prod(U));
   else
      X = -2*log(U);
   end
else         % odd degrees of freedom
   U = rand(k,n);
   Z = randn(1,n);
   if k ~= 1
      X = Z.^2-2*log(prod(U));
   else
      X = Z.^2-2*log(U);
   end
end
```

The use of this function to generate random variables is left as an exercise.
❑

The chi-square distribution is useful in situations where we need to systematically investigate the behavior of a statistic by changing the skewness of the distribution. As the degrees of freedom for a chi-square increases, the distribution changes from being right skewed to one approaching normality and symmetry.

### Beta

The beta distribution is useful in simulations because it covers a wide range of distribution shapes, depending on the values of the parameters $\alpha$ and $\beta$. These shapes include skewed, uniform, approximately normal, and a bimodal distribution with an interior dip.

First, we describe a simple approach for generating beta random variables with parameters $\alpha$ and $\beta$, when both are integers [Rubinstein, 1981; Gentle, 1998]. It is known [David, 1981] that the $k$ -th order statistic of $n$ uniform (0,1) variates is distributed according to a beta distribution with parameters $k$ and $n - k + 1$. This means that we can generate random variables from the beta distribution using the following procedure.

*PROCEDURE - BETA RANDOM VARIABLES (INTEGER PARAMETERS)*

1. Generate $\alpha + \beta - 1$ uniform random numbers: $U_1, \ldots, U_{\alpha+\beta-1}$
2. Deliver $X = U_{(\alpha)}$ which is the $\alpha$-th order statistic.

One simple way to generate random variates from the beta distribution is to use the following result from Rubinstein [1981]. If $Y_1$ and $Y_2$ are independent random variables, where $Y_1$ has a gamma distribution with parameters $\alpha$ and 1, and $Y_2$ follows a gamma distribution with parameters $\beta$ and 1, then

$$X = \frac{Y_1}{Y_1 + Y_2} \tag{4.17}$$

is from a beta distribution with parameters $\alpha$ and $\beta$. This is the method that is used in the MATLAB Statistics Toolbox function **betarnd** that generates random variates from the beta distribution. We illustrate the use of **betarnd** in the following example.

### Example 4.9

We use this example to illustrate the use of the MATLAB Statistics Toolbox function that generates beta random variables. In general, most of these toolbox functions for generating random variables use the following general syntax:

```
rvs = pdfrnd(par1,par2,nrow,ncol);
```

Here, ***pdf*** refers to the type of distribution (see Table 4.1, on page 106). The first several arguments represent the appropriate parameters of the distribution, so the number of them might change. The last two arguments denote the number of rows and the number of columns in the array of random variables that are returned by the function. We use the function **betarnd** to generate random variables from two beta distributions with different parameters $\alpha$ and $\beta$. First we look at the case where $\alpha = 3$ and $\beta = 3$. So, to generate $n = 500$ beta random variables (that are returned in a row vector), we use the following commands:

```
% Let a = 3, b = 3
n = 500;
a = 3;
b = 3;
rvs = betarnd(a,b,1,n);
```

We can construct a histogram of the random variables and compare it to the corresponding beta probability density function. This is easily accomplished in MATLAB as shown below.

```
% Now do the histogram.
[N,h] = hist(rvs,10);
% Change bar heights.
N = N/(h(2)-h(1))/n;
% Now get the theoretical probability density.
x = 0:.05:1;
y = betapdf(x,a,b);
plot(x,y)
axis equal
bar(h,N,1,'w')
hold on
plot(x,y,'k')
hold off
```

The result is shown in the left plot of Figure 4.6. Notice that this density looks approximately bell-shaped. The beta density on the right has parameters $\alpha = 0.5$ and $\beta = 0.5$. We see that this curve has a dip in the middle with modes on either end. The reader is asked to construct this plot in the exercises.

❑

### Multivariate Normal

In the following chapters, we will have many applications where we need to generate multivariate random variables in order to study the algorithms of computational statistics as they apply to multivariate distributions. Thus, we need some methods for generating multivariate random variables. The easi-

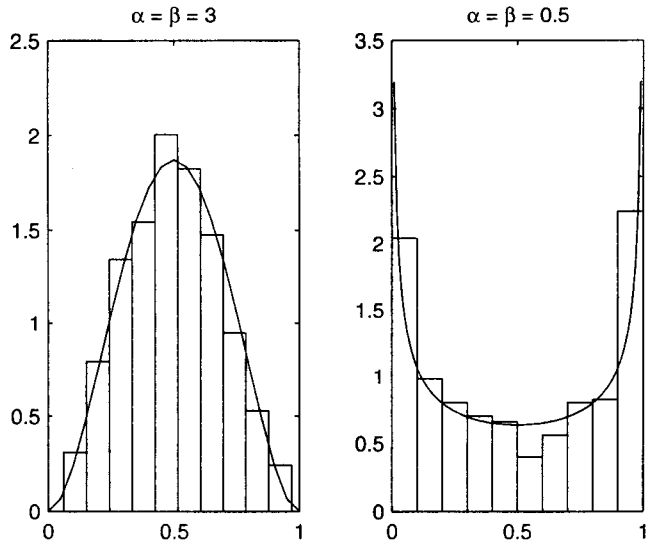



FIGURE 4.6
This figure shows two histograms created from random variables generated from the beta distribution. The beta distribution on the left has parameters $\alpha = 3$ and $\beta = 3$, while the one on the right has parameters $\alpha = 0.5$ and $\beta = 0.5$.

est distribution of this type to generate is the multivariate normal. We cover other methods for generating random variables from more general multivariate distributions in Chapter 11.

The method is similar to the one used to generate random variables from a univariate normal distribution. One starts with a $d$-dimensional vector of standard normal random numbers. These can be transformed to the desired distribution using

$$\mathbf{x} = \mathbf{R}^T\mathbf{z} + \boldsymbol{\mu}. \tag{4.18}$$

Here $\mathbf{z}$ is a $d \times 1$ vector of standard normal random numbers, $\boldsymbol{\mu}$ is a $d \times 1$ vector representing the mean, and $\mathbf{R}$ is a $d \times d$ matrix such that $\mathbf{R}^T\mathbf{R} = \Sigma$. The matrix $\mathbf{R}$ can be obtained in several ways, one of which is the Cholesky factorization of the covariance matrix $\Sigma$. This is the method we illustrate below. Another possibility is to factor the matrix using singular value decomposition, which will be shown in the examples provided in Chapter 5.

### Example 4.10

The function **csmvrnd** generates multivariate normal random variables using the Cholesky factorization. Note that we are transposing the transformation given in Equation 4.18, yielding the following

$$\mathbf{X} = \mathbf{ZR} + \mu^T,$$

where $\mathbf{X}$ is an $n \times d$ matrix of $d$-dimensional random variables and $\mathbf{Z}$ is an $n \times d$ matrix of standard normal random variables.

```
% function X = csmvrnd(mu,covm,n);
% This function will return n multivariate random
% normal variables with d-dimensional mean mu and
% covariance matrix covm. Note that the covariance
% matrix must be positive definite (all eigenvalues
% are greater than zero), and the mean
% vector is a column

function X = csmvrnd(mu,covm,n)
d = length(mu);
% Get Cholesky factorization of covariance.
R = chol(covm);
% Generate the standard normal random variables.
Z = randn(n,d);
X = Z*R + ones(n,1)*mu';
```

We illustrate its use by generating some multivariate normal random variables with $\mu^T = (-2, 3)$ and covariance

$$\Sigma = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}.$$

```
% Generate the multivariate random normal variables.
mu = [-2;3];
covm = [1 0.7 ; 0.7 1];
X = csmvrnd(mu,covm,500);
```

To check the results, we plot the random variables in a scatterplot in Figure 4.7. We can also calculate the sample mean and sample covariance matrix to compare with what we used as input arguments to **csmvrnd**. By typing **mean(X)** at the command line, we get

```
-2.0629    2.9394
```

Similarly, entering **corrcoef(X)** at the command line yields

```
1.0000    0.6957
0.6957    1.0000
```

We see that these values for the sample statistics correspond to the desired mean and covariance. We note that you could also use the **cov** function to compare the variances.
□

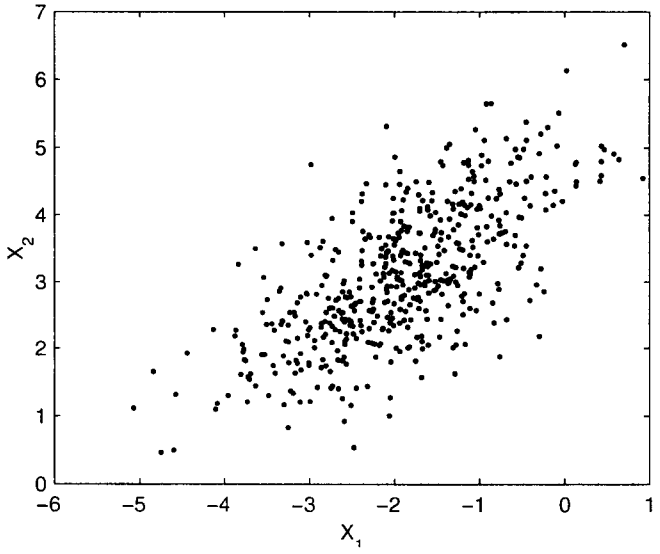



FIGURE 4.7
This shows the scatter plot of the random variables generated using the function **csmvrnd**.

### Generating Variates on a Sphere

In some applications, we would like to generate $d$-dimensional random variables that are distributed on the surface of the unit hypersphere $S^d$, $d = 2, \ldots$. Note that when $d = 2$ the surface is a circle, and for $d = 3$ the surface is a sphere. We will be using this technique in Chapter 5, where we present an algorithm for exploratory data analysis using projection pursuit. The easiest method is to generate $d$ standard normal random variables and then to scale them such that the magnitude of the vector is one. This is illustrated in the following example.

### Example 4.11

The following function **cssphrnd** generates random variables on a $d$-dimensional unit sphere. We illustrate its use by generating random variables that are on the unit circle $S^2$.

```
% function X = cssphrnd(n,d);
% This function will generate n d-dimensional
% random variates that are distributed on the
% unit d-dimensional sphere. d >= 2

function X = cssphrnd(n,d)
if d < 2
   error('ERROR - d must be greater than 1.')
   break
end
% Generate standard normal random variables.
tmp = randn(d,n);
% Find the magnitude of each column.
% Square each element, add and take the square root.
mag = sqrt(sum(tmp.^2));
% Make a diagonal matrix of them - inverses.
dm = diag(1./mag);
% Multiply to scale properly.
% Transpose so X contains the observations.
X = (tmp*dm)';
```

We can use this function to generate a set of random variables for $d = 2$ and plot the result in Figure 4.8.

```
X = cssphrnd(500,2);
plot(X(:,1),X(:,2),'x')
axis equal
xlabel('X_1'),ylabel('X_2')
```

❑

## 4.4 Generating Discrete Random Variables

### Binomial

A binomial random variable with parameters $n$ and $p$ represents the number of successes in $n$ independent trials. We can obtain a binomial random vari-

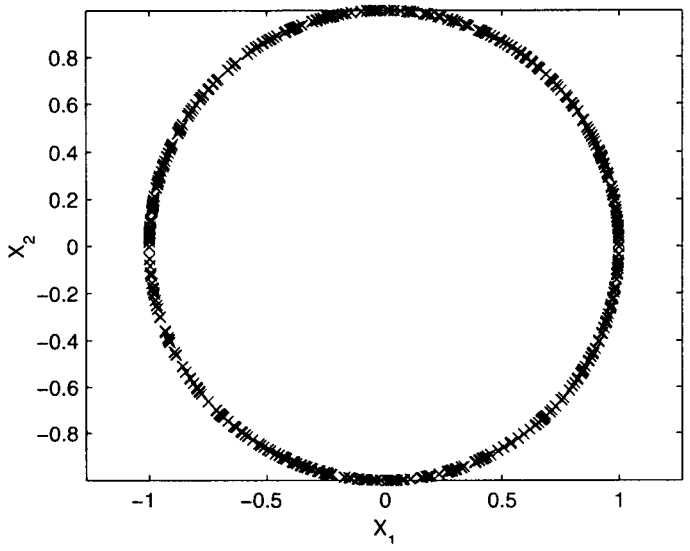



FIGURE 4.8
This is the scatter plot of the random variables generated in Example 4.11. These random variables are distributed on the surface of a 2-D unit sphere (i.e., a unit circle).

able by generating $n$ uniform random numbers $U_1, U_2, \ldots, U_n$ and letting $X$ be the number of $U_i$ that are less than or equal to $p$. This is easily implemented in MATLAB as illustrated in the following example.

## Example 4.12

We implement this algorithm for generating binomial random variables in the function **csbinrnd**.

```
% function X = csbinrnd(n,p,N)
% This function will generate N binomial
% random variables with parameters n and p.

function X = csbinrnd(n,p,N)
X = zeros(1,N);
% Generate the uniform random numbers:
% N variates of n trials.
U = rand(N,n);
% Loop over the rows, finding the number
% less than p
for i = 1:N
   ind = find(U(i,:) <= p);
   X(i) = length(ind);
```

```
end
```

We use this function to generate a set of random variables that are distributed according to the binomial distribution with parameters $n = 6$ and $p = 0.5$. The histogram of the random variables is shown in Figure 4.9. Before moving on, we offer the following more efficient way to generate binomial random variables in MATLAB:

```
X = sum(rand(n,N) <= p);
```

❑

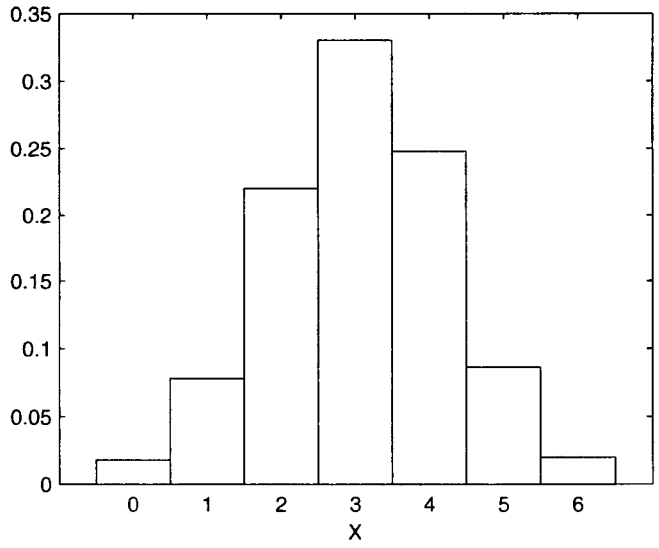



FIGURE 4.9
This is the histogram for the binomial random variables generated in Example 4.12. The parameters for the binomial are $n = 6$ and $p = 0.5$.

## Poisson

We use the inverse transform method for discrete random variables as described in Ross [1997] to generate variates from the Poisson distribution. We need the following recursive relationship between successive Poisson probabilities

$$p_{i+1} = P(X = i) = \frac{\lambda}{i+1} p_i; \qquad i \geq 0.$$

This leads to the following algorithm.

*PROCEDURE - GENERATING POISSON RANDOM VARIABLES*

1. Generate a uniform random number $U$.
2. Initialize the quantities: $i = 0$, $p_0 = e^{-\lambda}$, and $F_0 = p_0$.
3. If $U \leq F_i$, then deliver $X = i$. Return to step 1.
4. Else increment the values: $p_{i+1} = \lambda p_i/(i+1)$, $i = i+1$, and $F_{i+1} = F_i + p_{i+1}$.
5. Return to step 3.

This algorithm could be made more efficient when $\lambda$ is large. The interested reader is referred to Ross [1997] for more details.

### Example 4.13

The following shows how to implement the procedure for generating Poisson random variables in MATLAB.

```
% function X = cspoirnd(lam,n)
% This function will generate Poisson
% random variables with parameter lambda.
% The reference for this is Ross, 1997, page 50.

function x = cspoirnd(lam,n)
x = zeros(1,n);
j = 1;
while j <= n
   flag = 1;
   % initialize quantities
   u = rand(1);
   i = 0;
   p = exp(-lam);
   F = p;
   while flag % generate the variate needed
      if u <= F % then accept
         x(j) = i;
         flag = 0;
         j = j+1;
      else % move to next probability
         p = lam*p/(i+1);
         i = i+1;
         F = F + p;
      end
   end
```

```
end
```

We can use this to generate a set of Poisson random variables with $\lambda = 0.5$, and show a histogram of the data in Figure 4.10.

```
% Set the parameter for the Poisson.
lam = .5;
N = 500; % Sample size
x = cspoirnd(lam,N);
edges = 0:max(x);
f = histc(x,edges);
bar(edges,f/N,1,'w')
```

As an additional check to ensure that our algorithm is working correctly, we can determine the observed relative frequency of each value of the random variable $X$ and compare that to the corresponding theoretical values.

```
% Determine the observed relative frequencies.
% These are the estimated values.
relf = zeros(1,max(x)+1);
for i = 0:max(x)
   relf(i+1) = length(find(x==i))/N;
end
% Use the Statistics Toolbox function to get the
% theoretical values.
y = poisspdf(0:4,.5);
```

When we print these to the MATLAB command window, we have the following

```
% These are the estimated values.
relf = 0.5860    0.3080    0.0840    0.0200    0.0020
% These are the theoretical values.
y = 0.6065    0.3033    0.0758    0.0126    0.0016
```

❑

### Discrete Uniform

When we implement some of the Monte Carlo methods in Chapter 6 (such as the bootstrap), we will need the ability to generate numbers that follow the discrete uniform distribution. This is a distribution where $X$ takes on values in the set $\{1, 2, \ldots, N\}$, and the probability that $X$ equals any of the numbers is $1/N$. This distribution can be used to randomly sample without replacement from a group of $N$ objects.

We can generate from the discrete uniform distribution using the following transform

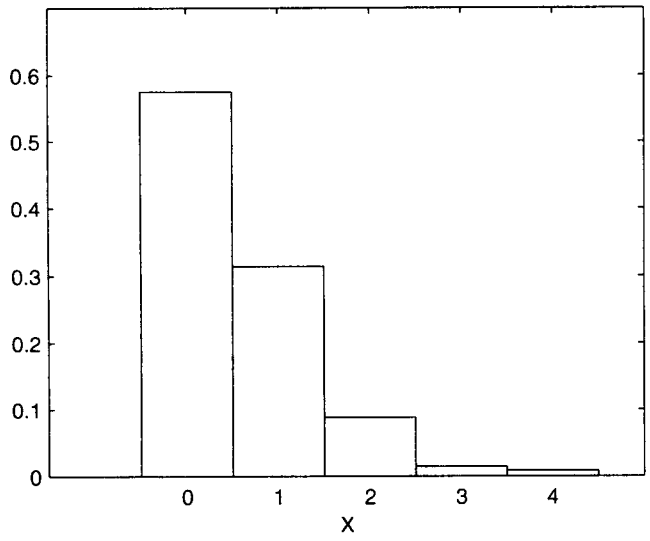



FIGURE 4.10
This is the histogram for random variables generated from the Poisson with $\lambda = 0.5$.

$$X = \lceil NU \rceil,$$

where the function $\lceil y \rceil$, $y \geq 0$ means to round up the argument $y$. The next example shows how to implement this in MATLAB.

## Example 4.14

The method for generating discrete uniform is implemented in the function **csdunrnd**, given below.

```
% function X = csdunrnd(N,n)
% This function will generate random variables
% from the discrete uniform distribution. It picks
% numbers uniformly between 1 and N.

function X = csdunrnd(N,n)
X = ceil(N*rand(1,n));
```

To verify that we are generating the right random variables, we can look at the observed relative frequencies. Each should have relative frequency of $1/N$.This is shown below where $N = 5$ and the sample size is 500.

```
N = 5;
n = 500;
x = csdunrnd(N,n);
```

```
% Determine the estimated relative frequencies.
relf = zeros(1,N);
for i = 1:N
   relf(i) = length(find(x==i))/n;
end
```

Printing out the observed relative frequencies, we have

```
relf = 0.1820    0.2080    0.2040    0.1900    0.2160
```

which is close to the theoretical value of $1/N = 1/5 = 0.2$.
□

## 4.5 MATLAB Code

The MATLAB Statistics Toolbox has functions that will generate random variables from all of the distributions discussed in Section 2.6. As we explained in that section, the analyst must keep in mind that probability distributions are often defined differently, so caution should be exercised when using any software package. Table 4.1 provides a partial list of the MATLAB functions that are available for random number generation. A complete list can be found in Appendix E. As before, the reader should note that the **gamrnd**, **weibrnd**, and **exprnd** functions use the alternative definition for the given distribution (see 24).

TABLE 4.1

Partial List of Functions in the MATLAB Statistics Toolbox for Generating Random Variables

| Distribution | MATLAB Function |
|---|---|
| Beta | **betarnd** |
| Binomial | **binornd** |
| Chi-Square | **chi2rnd** |
| Discrete Uniform | **unidrnd** |
| Exponential | **exprnd** |
| Gamma | **gamrnd** |
| Normal | **normrnd** |
| Poisson | **poissrnd** |
| Continuous Uniform | **unifrnd** |
| Weibull | **weibrnd** |

Another function that might prove useful in implementing computational statistics methods is called **randperm**. This is provided with the standard MATLAB software package, and it generates random permutations of the integers 1 to $n$. The result can be used to permute the elements of a vector. For example, to permute the elements of a vector **x** of size **n**, use the following MATLAB statements:

```
% Get the permuted indices.
ind = randperm(n);
% Now re-order based on the permuted indices.
xperm = x(ind);
```

We also provide some functions in the Computational Statistics Toolbox for generating random variables. These are outlined in Table 4.2. Note that these generate random variables using the distributions as defined in Chapter 2.

TABLE 4.2

List of Functions from Chapter 4 Included in the Computational Statistics Toolbox

| Distribution | MATLAB Function |
|---|---|
| Beta | csbetarnd |
| Binomial | csbinrnd |
| Chi-Square | cschirnd |
| Discrete Uniform | csdunrnd |
| Exponential | csexprnd |
| Gamma | csgamrnd |
| Multivariate Normal | csmvrnd |
| Poisson | cspoirnd |
| Points on a sphere | cssphrnd |

## 4.6 Further Reading

In this text we do not attempt to assess the computational efficiency of the methods for generating random variables. If the statistician or engineer is performing extensive Monte Carlo simulations, then the time it takes to generate random samples becomes important. In these situations, the reader is encouraged to consult Gentle [1998] or Rubinstein [1981] for efficient algorithms. Our goal is to provide methods that are easily implemented using MATLAB or other software, in case the data analyst must write his own functions for generating random variables from non-standard distributions.

There has been considerable research into methods for random number generation, and we refer the reader to the sources mentioned below for more information on the theoretical foundations. The book by Ross [1997] is an excellent resource and is suitable for advanced undergraduate students. He addresses simulation in general and includes a discussion of discrete event simulation and Markov chain Monte Carlo methods. Another text that covers the topic of random number generation and Monte Carlo simulation is Gentle [1998]. This book includes an extensive discussion of uniform random number generation and covers more advanced topics such as Gibbs sampling. Two other resources on random number generation are Rubinstein [1981] and Kalos and Whitlock [1986]. For a description of methods for generating random variables from more general multivariate distributions, see Johnson [1987]. The article by Deng and Lin [2000] offers improvements on some of the standard uniform random number generators.

A recent article in the MATLAB *News & Notes* [Spring, 2001] describes the method employed in MATLAB for obtaining normally distributed random variables. The algorithm that MATLAB uses for generating uniform random numbers is described in a similar newsletter article and is available for download at:

**www.mathworks.com/company/newsletter/pdf/Cleve.pdf**.

## Exercises

4.1. Repeat Example 4.3 using larger sample sizes. What happens to the estimated probability mass function (i.e., the relative frequencies from the random samples) as the sample size gets bigger?

4.2. Write the MATLAB code to implement Example 4.5. Generate 500 random variables from this distribution and construct a histogram (**hist** function) to verify your code.

4.3. Using the algorithm implemented in Example 4.3, write a MATLAB function that will take any probability mass function (i.e., a vector of probabilities) and return the desired number of random variables generated according to that probability function.

4.4. Write a MATLAB function that will return random numbers that are uniformly distributed over the interval $(a, b)$.

4.5. Write a MATLAB function that will return random numbers from the normal distribution with mean $\mu$ and variance $\sigma^2$. The user should be able to set values for the mean and variance as input arguments.

4.6. Write a function that will generate chi-square random variables with $\nu$ degrees of freedom by generating $\nu$ standard normals, squaring them and then adding them up. This uses the fact that

$$X = Z_1^2 + \ldots + Z_\nu^2$$

is chi-square with $\nu$ degrees of freedom. Generate some random variables and plot in a histogram. The degrees of freedom should be an input argument set by the user.

4.7. An alternative method for generating beta random variables is described in Rubinstein [1981]. Generate two variates $Y_1 = U_1^{1/\alpha}$ and $Y_2 = U_2^{1/\beta}$, where the $U_i$ are from the uniform distribution. If $Y_1 + Y_2 \leq 1$, then

$$X = \frac{Y_1}{Y_1 + Y_2},$$

is from a beta distribution with parameters $\alpha$ and $\beta$. Implement this algorithm.

4.8. Run Example 4.4 and generate 1000 random variables. Determine the number of variates that were rejected and the total number generated to obtain the random sample. What percentage were rejected? How efficient was it?

4.9. Run Example 4.4 and generate 500 random variables. Plot a histogram of the variates. Does it match the probability density function shown in Figure 4.3?

4.10. Implement Example 4.5 in MATLAB. Generate 100 random variables. What is the relative frequency of each value of the random variable $1, \ldots, 5$? Does this match the probability mass function?

4.11. Generate four sets of random variables with $\nu = 2, 5, 15, 20$, using the function **cschirnd**. Create histograms for each sample. How does the shape of the distribution depend on the degrees of freedom $\nu$?

4.12. Repeat Example 4.13 for larger sample sizes. Is the agreement better between the observed relative frequencies and the theoretical values?

4.13. Generate 1000 binomial random variables for $n = 5$ and $p = 0.3, 0.5, 0.8$. In each case, determine the observed relative frequencies and the corresponding theoretical probabilities. How is the agreement between them?

4.14. The MATLAB Statistics Toolbox has a GUI called **randtool**. This is an interactive demo that generates random variables from distributions that are available in the toolbox. The user can change parameter values and see the results via a histogram. There are options to change the sample size and to output the results. To start the GUI, simply type **randtool** at the command line. Run the function and experiment with the distributions that are discussed in the text (normal, exponential, gamma, beta, etc.).

4.15. The plot on the right in Figure 4.6 shows a histogram of beta random variables with parameters $\alpha = \beta = 0.5$. Construct a similar plot using the information in Example 4.9.

# Chapter 5

## *Exploratory Data Analysis*

### 5.1 Introduction

Exploratory data analysis (EDA) is quantitative detective work according to John Tukey [1977]. EDA is the philosophy that data should first be explored without assumptions about probabilistic models, error distributions, number of groups, relationships between the variables, etc. for the purpose of discovering what they can tell us about the phenomena we are investigating. The goal of EDA is to explore the data to reveal patterns and features that will help the analyst better understand, analyze and model the data. With the advent of powerful desktop computers and high resolution graphics capabilities, these methods and techniques are within the reach of every statistician, engineer and data analyst.

EDA is a collection of techniques for revealing information about the data and methods for visualizing them to see what they can tell us about the underlying process that generated it. In most situations, exploratory data analysis should precede confirmatory analysis (e.g., hypothesis testing, ANOVA, etc.) to ensure that the analysis is appropriate for the data set. Some examples and goals of EDA are given below to help motivate the reader.

- If we have a time series, then we would plot the values over time to look for patterns such as trends, seasonal effects or change points. In Chapter 11, we have an example of a time series that shows evidence of a change point in a Poisson process.
- We have observations that relate two characteristics or variables, and we are interested in how they are related. Is there a linear or a nonlinear relationship? Are there patterns that can provide insight into the process that relates the variables? We will see examples of this application in Chapters 7 and 10.
- We need to provide some summary statistics that describe the data set. We should look for outliers or aberrant observations that might contaminate the results. If EDA indicates extreme observations are

in the data set, then robust statistical methods might be more appropriate. In Chapter 10, we illustrate an example where a graphical look at the data indicates the presence of outliers, so we use a robust method of nonparametric regression.

- We have a random sample that will be used to develop a model. This model will be included in our simulation of a process (e.g., simulating a physical process such as a queue). We can use EDA techniques to help us determine how the data might be distributed and what model might be appropriate.

In this chapter, we will be discussing graphical EDA and how these techniques can be used to gain information and insights about the data. Some experts include techniques such as smoothing, probability density estimation, clustering and principal component analysis in exploratory data analysis. We agree that these can be part of EDA, but we do not cover them in this chapter. Smoothing techniques are discussed in Chapter 10 where we present methods for nonparametric regression. Techniques for probability density estimation are presented in Chapter 8, but we do discuss simple histograms in this chapter. Methods for clustering are described in Chapter 9. Principal component analysis is not covered in this book, because the subject is discussed in many linear algebra texts [Strang, 1988; Jackson, 1991].

It is likely that some of the visualization methods in this chapter are familiar to statisticians, data analysts and engineers. As we stated in Chapter 1, one of the goals of this book is to promote the use of MATLAB for statistical analysis. Some readers might not be familiar with the extensive graphics capabilities of MATLAB, so we endeavor to describe the most useful ones for data analysis. In Section 5.2, we consider techniques for visualizing univariate data. These include such methods as stem-and-leaf plots, box plots, histograms, and quantile plots. We turn our attention to techniques for visualizing bivariate data in Section 5.3 and include a description of surface plots, scatterplots and bivariate histograms. Section 5.4 offers several methods for viewing multi-dimensional data, such as slices, isosurfaces, star plots, parallel coordinates, Andrews curves, projection pursuit, and the grand tour.

## 5.2 Exploring Univariate Data

Two important goals of EDA are: 1) to determine a reasonable model for the process that generated the data, and 2) to locate possible outliers in the sample. For example, we might be interested in finding out whether the distribution that generated the data is symmetric or skewed. We might also like to know whether it has one mode or many modes. The univariate visualization techniques presented here will help us answer questions such as these.

### Histograms

A ***histogram*** is a way to graphically represent the frequency distribution of a data set. Histograms are a good way to

- summarize a data set to understand general characteristics of the distribution such as shape, spread or location,
- suggest possible probabilistic models, or
- determine unusual behavior.

In this chapter, we look only at the simple, basic histogram. Variants and extensions of the histogram are discussed in Chapter 8.

A ***frequency histogram*** is obtained by creating a set of bins or intervals that cover the range of the data set. It is important that these bins do not overlap and that they have equal width. We then count the number of observations that fall into each bin. To visualize this, we plot the frequency as the height of a bar, with the width of the bar representing the width of the bin. The histogram is determined by two parameters, the bin width and the starting point of the first bin. We discuss these issues in greater detail in Chapter 8. ***Relative frequency histograms*** are obtained by representing the height of the bin by the relative frequency of the observations that fall into the bin.

The basic MATLAB package has a function for calculating and plotting a univariate histogram. This function is illustrated in the example given below.

### Example 5.1

In this example, we look at a histogram of the data in **forearm**. These data [Hand, et al., 1994; Pearson and Lee, 1903] consist of 140 measurements of the length in inches of the forearm of adult males. We can obtain a simple histogram in MATLAB using these commands:

```
load forearm
subplot(1,2,1)
% The hist function optionally returns the
% bin centers and frequencies.
[n,x] = hist(forearm);
% Plot and use the argument of width=1
% to produce bars that touch.
bar(x,n,1);
axis square
title('Frequency Histogram')
% Now create a relative frequency histogram.
% Divide each box by the total number of points.
subplot(1,2,2)
bar(x,n/140,1)
title('Relative Frequency Histogram')
axis square
```

These plots are shown in Figure 5.1. Notice that the shapes of the histograms are the same in both types of histograms, but the vertical axis is different. From the shape of the histograms, it seems reasonable to assume that the data are normally distributed.

□

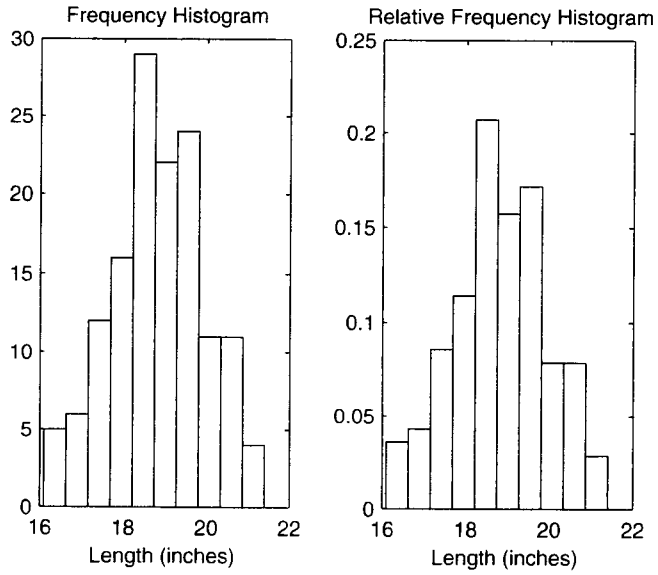



FIGURE 5.1
On the left is a frequency histogram of the `forearm` data, and on the right is the relative frequency histogram. These indicate that the distribution is unimodal and that the normal distribution is a reasonable model.

One problem with using a frequency or relative frequency histogram is that they do not represent meaningful probability densities, because they do not integrate to one. This can be seen by superimposing a corresponding normal distribution over the relative frequency histogram as shown in Figure 5.2.

A ***density histogram*** is a histogram that has been normalized so it will integrate to one. That means that if we add up the *areas* represented by the bars, then they should add up to one. A density histogram is given by the following equation

$$\hat{f}(x) = \frac{\nu_k}{nh} \qquad x \text{ in } B_k, \tag{5.1}$$

where $B_k$ denotes the $k$-th bin, $\nu_k$ represents the number of data points that fall into the $k$-th bin and $h$ represents the width of the bins. In the following

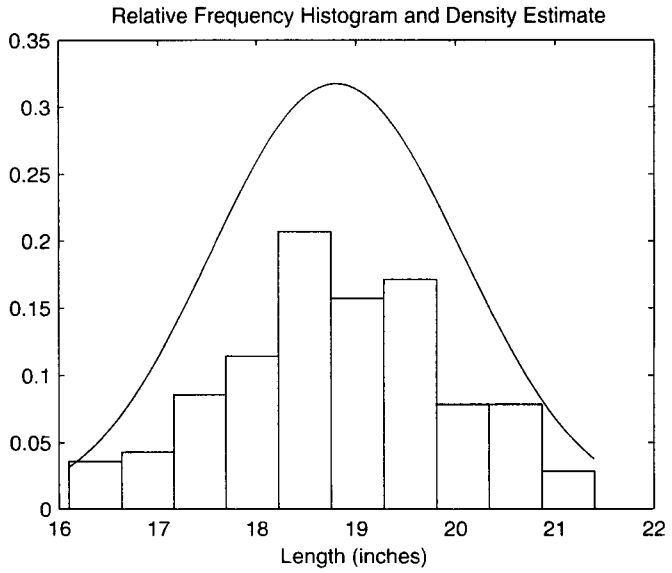



FIGURE 5.2
This shows a relative frequency histogram of the **forearm** data. Superimposed on the histogram is the normal probability density function using parameters estimated from the data. Note that the curve is higher than the histogram, indicating that the histogram is not a valid probability density function.

example, we reproduce the histogram of Figure 5.2 using the density histogram.

## Example 5.2

Here we explore the **forearm** data using a density histogram. Assuming a normal distribution and estimating the parameters from the data, we can superimpose a smooth curve that represents an estimated density for the normal distribution.

```
% Get parameter estimates for the normal distribution.
mu = mean(forearm);
v = var(forearm);
% Obtain normal pdf based on parameter estimates.
xp = linspace(min(forearm),max(forearm));
yp = normp(xp,mu,v);
% Get the information needed for a histogram.
[nu,x] = hist(forearm);
% Get the widths of the bins.
h = x(2)-x(1);
```

```
% Plot as density histogram - Equation 5.1.
bar(x,nu/(140*h),1)
hold on
plot(xp,yp)
xlabel('Length (inches)')
title('Density Histogram and Density Estimate')
hold off
```

The results are shown in Figure 5.3. Note that the assumption of normality for the data is not unreasonable. The estimated density function and the density histogram match up quite well.
❑

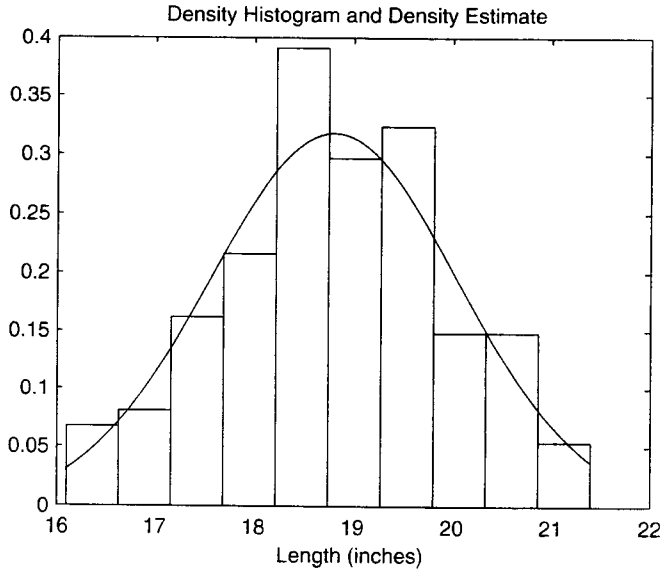



FIGURE 5.3
Density histogram for the `forearm` data. The curve represents a normal probability density function with parameters given by the sample mean and sample variance of the data. From this we see that the normal distribution is a reasonable probabilistic model.

### Stem-and-Leaf

Stem-and-leaf plots were introduced by Tukey [1977] as a way of displaying data in a structured list. Presenting data in a table or an ordered list does not readily convey information about how the data are distributed, as is the case with histograms.

If we have data where each observation consists of at least two digits, then we can construct a stem-and-leaf diagram. To display these, we separate each measurement into two parts: the ***stem*** and the ***leaf***. The stems are comprised of the leading digit or digits, and the remaining digit makes up the leaf. For example, if we had the number 75, then the stem is the 7, and the leaf is the 5. If the number is 203, then the stem is 20 and the leaf is 3.

The stems are listed to the left of a vertical line with all of the leaves corresponding to that stem listed to the right. If the data contain decimal places, then they can be rounded for easier display. An alternative is to move the decimal place to specify the appropriate leaf unit. We provide a function with the text that will construct stem-and-leaf plots, and its use is illustrated in the next example.

### Example 5.3

The heights of 32 Tibetan skulls [Hand, et al. 1994; Morant, 1923] measured in millimeters is given in the file **tibetan**. These data comprise two groups of skulls collected in Tibet. One group of 17 skulls comes from graves in Sikkim and nearby areas of Tibet and the other 15 skulls come from a battlefield in Lhasa. The original data contain five measurements, but for this example, we only use the fourth measurement. This is the upper face height, and we round to the nearest millimeter. We use the function **csstemleaf** that is provided with the text.

```
load tibetan
% This loads up all 5 measurements of the skulls.
% We use the fourth characteristic to illustrate
% the stem-and-leaf plot. We first round them.
x = round(tibetan(:,4));
csstemleaf(x)
title('Height (mm) of Tibetan Skulls')
```

The resulting stem-and-leaf is shown in Figure 5.4. From this plot, we see there is not much evidence that there are two groups of skulls, if we look only at the characteristic of upper face height. We will explore these data further in Chapter 9, where we apply pattern recognition methods to the problem. ❑

It is possible that we do not see much evidence for two groups of skulls because there are too few stems. EDA is an iterative process, where the analyst should try several visualization methods in search of patterns and information in the data. An alternative approach is to plot more than one line per stem. The function **csstemleaf** has an optional argument that allows the user to specify two lines per stem. The default value is one line per stem, as we saw in Example 5.3. When we plot two lines per stem, leaves that correspond to the digits 0 through 4 are plotted on the first line and those that have digits 5 through 9 are shown on the second line. A stem-and-leaf with two lines per stem for the Tibetan skull data is shown in Figure 5.5. In practice,

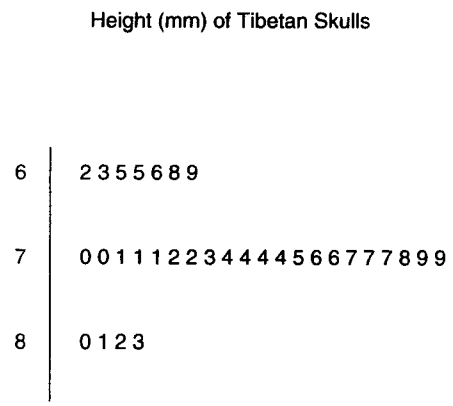



**FIGURE 5.4**
This shows the stem-and-leaf plot for the upper face height of 32 Tibetan skulls. The data have been rounded to the nearest millimeter.

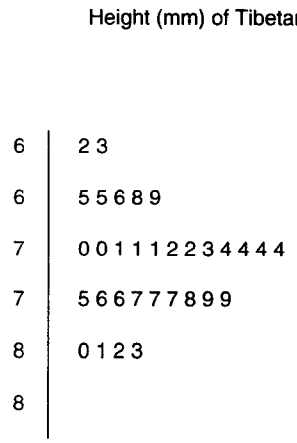



**FIGURE 5.5**
This shows a stem-and-leaf plot for the upper face height of 32 Tibetan skulls where we now have two lines per stem. Note that we see approximately the same information (a unimodal distribution) as in Figure 5.4.

one could plot a stem-and-leaf with one and with two lines per stem as a way of discovering more about the data. The stem-and-leaf is useful in that it approximates the shape of the density, and it also provides a listing of the data. One can usually recover the original data set from the stem-and-leaf (if it has not been rounded), unlike the histogram. A disadvantage of the stem-and-leaf plot is that it is not useful for large data sets, while a histogram is very effective in reducing and displaying massive data sets.

### Quantile-Based Plots - Continuous Distributions

If we need to compare two distributions, then we can use the quantile plot to visually compare them. This is also applicable when we want to compare a distribution and a sample or to compare two samples. In comparing the distributions or samples, we are interested in knowing how they are shifted relative to each other. In essence, we want to know if they are distributed in the same way. This is important when we are trying to determine the distribution that generated our data, possibly with the goal of using that information to generate data for Monte Carlo simulation. Another application where this is useful is in checking model assumptions, such as normality, before we conduct our analysis.

In this part, we discuss several versions of quantile-based plots. These include ***quantile-quantile plots*** (q-q plots) and ***quantile plots*** (sometimes called a ***probability plot***). Quantile plots for discrete data are discussed next. The quantile plot is used to compare a sample with a theoretical distribution. Typically, a q-q plot (sometimes called an ***empirical quantile plot***) is used to determine whether two random samples are generated by the same distribution. It should be noted that the q-q plot can also be used to compare a random sample with a theoretical distribution by generating a sample from the theoretical distribution as the second sample.

#### *Q-Q Plot*

The q-q plot was originally proposed by Wilk and Gnanadesikan [1968] to visually compare two distributions by graphing the quantiles of one versus the quantiles of the other. Say we have two data sets consisting of univariate measurements. We denote the order statistics for the first data set by

$$x_{(1)}, x_{(2)}, \ldots, x_{(n)}.$$

Let the order statistics for the second data set be

$$y_{(1)}, y_{(2)}, \ldots, y_{(m)},$$

with $m \leq n$.

We look first at the case where the sizes of the data sets are equal, so $m = n$. In this case, we plot as points the sample quantiles of one data set versus the other data set. This is illustrated in Example 5.4. If the data sets come from the same distribution, then we would expect the points to approximately follow a straight line.

A major strength of the quantile-based plots is that they do not require the two samples (or the sample and theoretical distribution) to have the same location and scale parameter. If the distributions are the same, but differ in location or scale, then we would still expect the quantile-based plot to produce a straight line.

### Example 5.4

We will generate two sets of normal random variables and construct a q-q plot. As expected, the q-q plot (Figure 5.6) follows a straight line, indicating that the samples come from the same distribution.

```
% Generate the random variables.
x = randn(1,75);
y = randn(1,75);
% Find the order statistics.
xs = sort(x);
ys = sort(y);
% Now construct the q-q plot.
plot(xs,ys,'o')
xlabel('X - Standard Normal')
ylabel('Y - Standard Normal')
axis equal
```

If we repeat the above MATLAB commands using a data set generated from an exponential distribution and one that is generated from the standard normal, then we have the plot shown in Figure 5.7. Note that the points in this q-q plot do not follow a straight line, leading us to conclude that the data are not generated from the same distribution.

❑

We now look at the case where the sample sizes are not equal. Without loss of generality, we assume that $m < n$. To obtain the q-q plot, we graph the $y_{(i)}$, $i = 1, \ldots, m$ against the $(i - 0.5)/m$ quantile of the other data set. Note that this definition is not unique [Cleveland, 1993]. The $(i - 0.5)/m$ quantiles of the $x$ data are usually obtained via interpolation, and we show in the next example how to use the function **csquantiles** to get the desired plot.

Users should be aware that q-q plots provide a rough idea of how similar the distribution is between two random samples. If the sample sizes are small, then a lot of variation is expected, so comparisons might be suspect. To help aid the visual comparison, some q-q plots include a reference line. These are lines that are estimated using the first and third quartiles $(q_{0.25}, q_{0.75})$ of each data set and extending the line to cover the range of the data. The

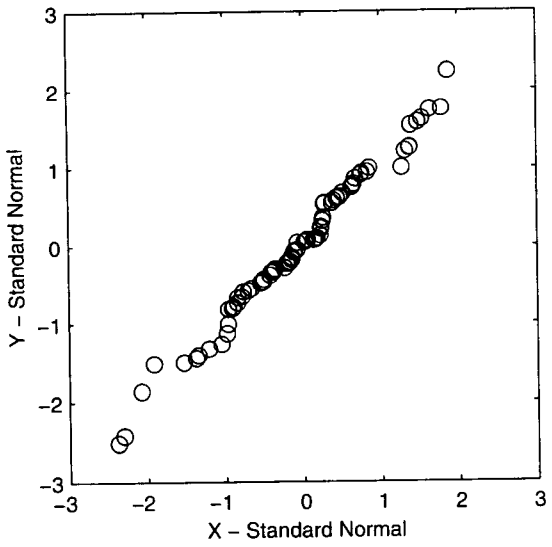



FIGURE 5.6
This is a q-q plot of $x$ and $y$ where both data sets are generated from a standard normal distribution. Note that the points follow a line, as expected.

MATLAB Statistics Toolbox provides a function called **qqplot** that displays this type of plot. We show below how to add the reference line.

## Example 5.5

This example shows how to do a q-q plot when the samples do not have the same number of points. We use the function **csquantiles** to get the required sample quantiles from the data set that has the larger sample size. We then plot these versus the order statistics of the other sample, as we did in the previous examples. Note that we add a reference line based on the first and third quartiles of each data set, using the function **polyfit** (see Chapter 7 for more information on this function).

```
% Generate the random variables.
m = 50;
n = 75;
x = randn(1,n);
y = randn(1,m);
% Find the order statistics for y.
ys = sort(y);
% Now find the associated quantiles using the x.
% Probabilities for quantiles:
p = ((1:m) - 0.5)/m;
```

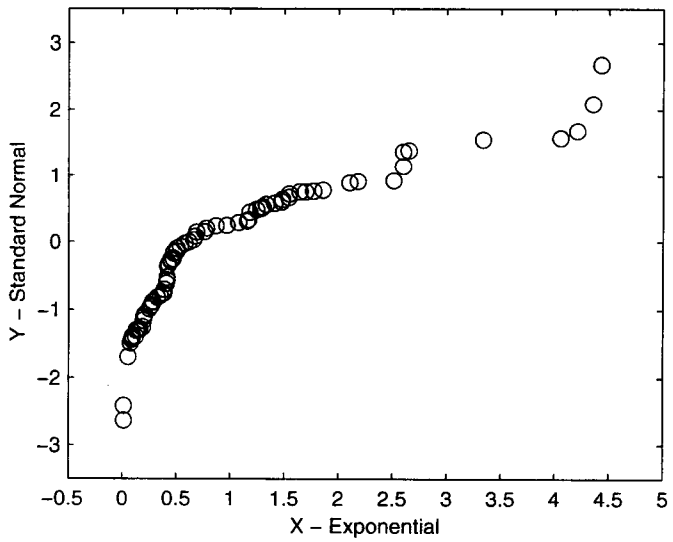



FIGURE 5.7
This is a q-q plot where one random sample is generated from the exponential distribution and one is generated by a standard normal distribution. Note that the points do not follow a straight line, indicating that the distributions that generated the random variables are not the same.

```
xs = csquantiles(x,p);
% Construct the plot.
plot(xs,ys,'ko')
% Get the reference line.
% Use the 1st and 3rd quartiles of each set to
% get a line.
qy = csquantiles(y,[0.25,0.75]);
qx = csquantiles(x,[0.25,0.75]);
[pol, s] = polyfit(qx,qy,1);
% Add the line to the figure.
yhat = polyval(pol,xs);
hold on
plot(xs,yhat,'k')
xlabel('Sample Quantiles - X'),
ylabel('Sorted Y Values')
hold off
```

From Figure 5.8, the assumption that each data set is generated according to the same distribution seems reasonable.

□

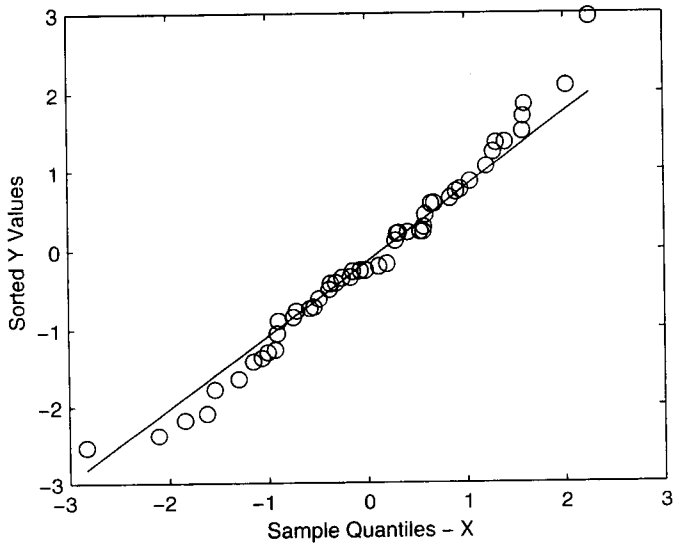



FIGURE 5.8
Here we show the q-q plot of Example 5.5. In this example, we also show the reference line estimated from the first and third quartiles. The q-q plot shows that the data do seem to come from the same distribution.

### *Quantile Plots*

A ***quantile plot*** or ***probability plot*** is one where the theoretical quantiles are plotted against the order statistics for the sample. Thus, on one axis we plot the $x_{(i)}$ and on the other axis we plot

$$F^{-1}\left(\frac{i-0.5}{n}\right),$$

where $F^{-1}(.)$ denotes the inverse of the cumulative distribution function for the hypothesized distribution. As before, the 0.5 in the above argument can be different [Cleveland, 1993]. A well-known example of a quantile plot is the ***normal probability plot***, where the ordered sample versus the quantiles of the normal distribution are plotted.

The MATLAB Statistics Toolbox has two functions for obtaining quantile plots. One is called **`normplot`**, and it produces a normal probability plot. So, if one would like to assess the assumption that a data set comes from a normal distribution, then this is the one to use. There is also a function for constructing a quantile plot that compares a data set to the Weibull distribution. This is called **`weibplot`**. For quantile plots with other theoretical distribu-

tions, one can use the MATLAB code given below, substituting the appropriate function to get the theoretical quantiles.

### Example 5.6

This example illustrates how you can display a quantile plot in MATLAB. We first generate a random sample from the standard normal distribution as our data set. The sorted sample is an estimate of the $(i-0.5)/n$ quantile, so we next calculate these probabilities and get the corresponding theoretical quantiles. Finally, we use the function **norminv** from the Statistics Toolbox to get the theoretical quantiles for the normal distribution. The resulting quantile plot is shown in Figure 5.9.

```
% Generate a random sample from a standard normal.
x = randn(1,100);
% Get the probabilities.
prob = ((1:100)-0.5)/100;
% Now get the theoretical quantiles.
qp = norminv(prob,0,1);
% Now plot theoretical quantiles versus
% the sorted data.
plot(sort(x),qp,'ko')
xlabel('Sorted Data')
ylabel('Standard Normal Quantiles')
```

To further illustrate these concepts, let's see what happens when we generate a random sample from a uniform $(0, 1)$ distribution and check it against the normal distribution. The MATLAB code is given below, and the quantile plot is shown in Figure 5.10. As expected, the points do not lie on a line, and we see that the data are not from a normal distribution.

```
% Generate a random sample from a
% uniform distribution.
x = rand(1,100);
% Get the probabilities.
prob = ((1:100)-0.5)/100;
% Now get the theoretical quantiles.
qp = norminv(prob,0,1);
% Now plot theoretical quantiles versus
% the sorted data.
plot(sort(x),qp,'ko')
ylabel('Standard Normal Quantiles')
xlabel('Sorted Data')
```

□

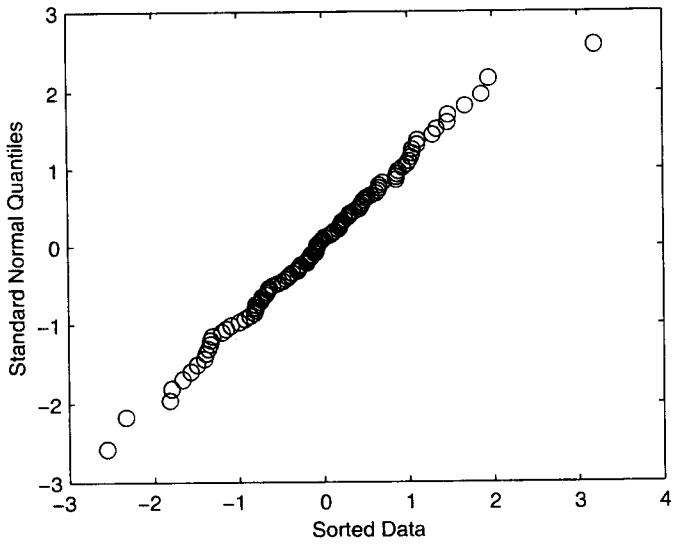



FIGURE 5.9
This is a quantile plot or normal probability plot of a random sample generated from a standard normal distribution. Note that the points approximately follow a straight line, indicating that the normal distribution is a reasonable model for the sample.

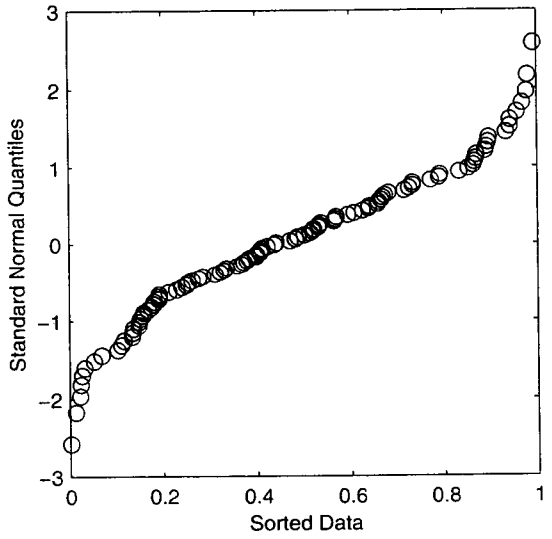



FIGURE 5.10
Here we have a quantile plot where the sample is generated from a uniform distribution, and the theoretical quantiles are from the normal distribution. The shape of the curve verifies that the sample is not from a normal distribution.

### Quantile Plots - Discrete Distributions

Previously, we discussed quantile plots that are primarily used for continuous data. We would like to have a similar technique for graphically comparing the shapes of discrete distributions. Hoaglin and Tukey [1985] developed several plots to accomplish this. We present two of them here: the ***Poissonness plot*** and the ***binomialness plot***. These will enable us to search for evidence that our discrete data follow a Poisson or a binomial distribution. They also serve to highlight which points might be incompatible with the model.

#### *Poissonness Plot*

Typically, discrete data are whole number values that are often obtained by counting the number of times something occurs. For example, these might be the number of traffic fatalities, the number of school-age children in a household, the number of defects on a hard drive, or the number of errors in a computer program. We sometimes have the data in the form of a frequency distribution that lists the possible count values (e.g., 0.1, 2, ... ) and the number of observations that are equal to the count values.

The counts will be denoted as $k$, with $k = 0, 1, \ldots, L$. We will assume that $L$ is the maximum observed value for our discrete variable or counts in the data set and that we are interested in all counts between 0 and $L$. Thus, the total number of observations in the sample is

$$N = \sum_{k=0}^{L} n_k ,$$

where $n_k$ represents the number of observations that are equal to the count $k$.

A basic Poissonness plot is constructed by plotting the count values $k$ on the horizontal axis and

$$\varphi(n_k) = \ln(k! n_k / N) \tag{5.2}$$

on the vertical axis. These are plotted as symbols, similar to the quantile plot. If a Poisson distribution is a reasonable model for the data, then this should follow a straight line. Systematic curvature in the plot would indicate that these data are not consistent with a Poisson distribution. The values for $\varphi(n_k)$ tend to have more variability when $n_k$ is small, so Hoaglin and Tukey [1985] suggest plotting a special symbol or a '1' to highlight these points.

### Example 5.7

This example is taken from Hoaglin and Tukey [1985]. In the late 1700's, Alexander Hamilton, John Jay and James Madison wrote a series of 77 essays under the title of *The Federalist*. These appeared in the newspapers under a

### Quantile Plots - Discrete Distributions

Previously, we discussed quantile plots that are primarily used for continuous data. We would like to have a similar technique for graphically comparing the shapes of discrete distributions. Hoaglin and Tukey [1985] developed several plots to accomplish this. We present two of them here: the ***Poissonness plot*** and the ***binomialness plot***. These will enable us to search for evidence that our discrete data follow a Poisson or a binomial distribution. They also serve to highlight which points might be incompatible with the model.

#### *Poissonness Plot*

Typically, discrete data are whole number values that are often obtained by counting the number of times something occurs. For example, these might be the number of traffic fatalities, the number of school-age children in a household, the number of defects on a hard drive, or the number of errors in a computer program. We sometimes have the data in the form of a frequency distribution that lists the possible count values (e.g., 0.1, 2, ... ) and the number of observations that are equal to the count values.

The counts will be denoted as $k$, with $k = 0, 1, \ldots, L$. We will assume that $L$ is the maximum observed value for our discrete variable or counts in the data set and that we are interested in all counts between 0 and $L$. Thus, the total number of observations in the sample is

$$N = \sum_{k=0}^{L} n_k,$$

where $n_k$ represents the number of observations that are equal to the count $k$.

A basic Poissonness plot is constructed by plotting the count values $k$ on the horizontal axis and

$$\varphi(n_k) = \ln(k! n_k / N) \tag{5.2}$$

on the vertical axis. These are plotted as symbols, similar to the quantile plot. If a Poisson distribution is a reasonable model for the data, then this should follow a straight line. Systematic curvature in the plot would indicate that these data are not consistent with a Poisson distribution. The values for $\varphi(n_k)$ tend to have more variability when $n_k$ is small, so Hoaglin and Tukey [1985] suggest plotting a special symbol or a '1' to highlight these points.

### Example 5.7

This example is taken from Hoaglin and Tukey [1985]. In the late 1700's, Alexander Hamilton, John Jay and James Madison wrote a series of 77 essays under the title of *The Federalist*. These appeared in the newspapers under a

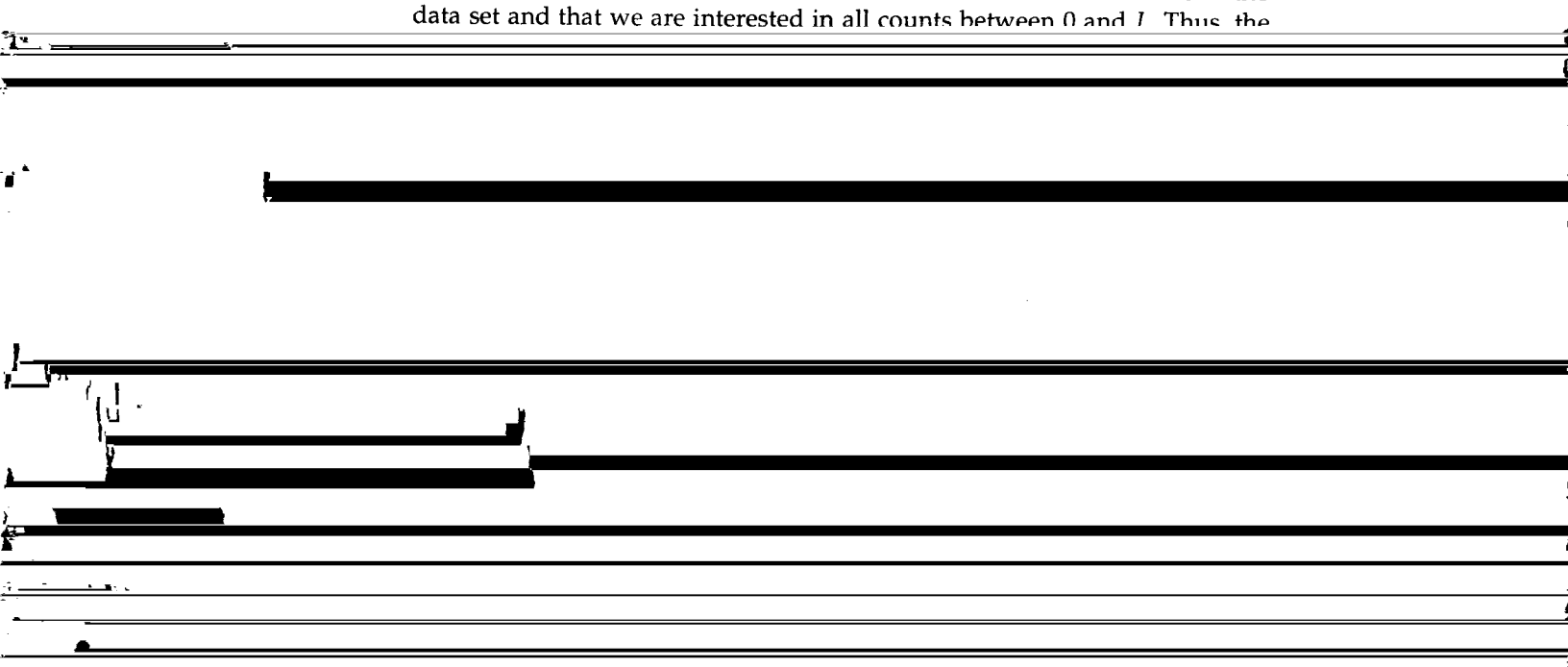
### Quantile Plots - Discrete Distributions

Previously, we discussed quantile plots that are primarily used for continuous data. We would like to have a similar technique for graphically comparing the shapes of discrete distributions. Hoaglin and Tukey [1985] developed several plots to accomplish this. We present two of them here: the ***Poissonness plot*** and the ***binomialness plot***. These will enable us to search for evidence that our discrete data follow a Poisson or a binomial distribution. They also serve to highlight which points might be incompatible with the model.

#### *Poissonness Plot*

Typically, discrete data are whole number values that are often obtained by counting the number of times something occurs. For example, these might be the number of traffic fatalities, the number of school-age children in a household, the number of defects on a hard drive, or the number of errors in a computer program. We sometimes have the data in the form of a frequency distribution that lists the possible count values (e.g., $0, 1, 2, \ldots$) and the number of observations that are equal to the count values.

The counts will be denoted as $k$, with $k = 0, 1, \ldots, L$. We will assume that $L$ is the maximum observed value for our discrete variable or counts in the data set and that we are interested in all counts between 0 and $L$. Thus, the



of the significance of the differences between the values. If the notches do not overlap, then there is evidence that the medians are significantly different. The length of the whisker is easily adjusted using optional input arguments to `boxplot`. For more information on this function and to find out what other options are available, type `help boxplot` at the MATLAB command line.

### Example 5.10

In this example, we first generate random variables from a uniform distribution on the interval $(0, 1)$, a standard normal distribution, and an exponential distribution. We will then display the box plots corresponding to each sample using the MATLAB function `boxplot`.

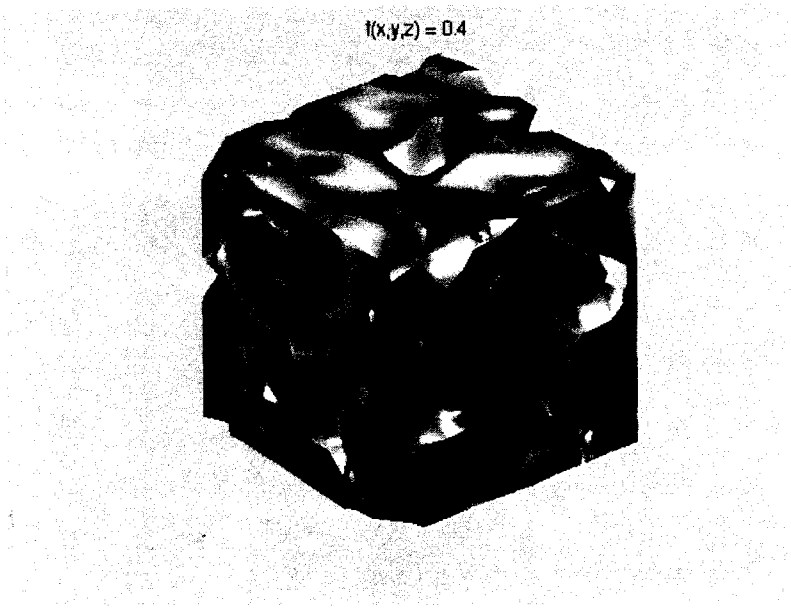



FIGURE 5.32
This is the **isosurface** of Figure 5.30 with **isocaps** added. Note that the color of the edges is mapped to the volume. The default is to map all values above $f(x, y, z) = 0.4$ to the color on the **isocaps**. This can be changed by an input argument to **isocaps**.

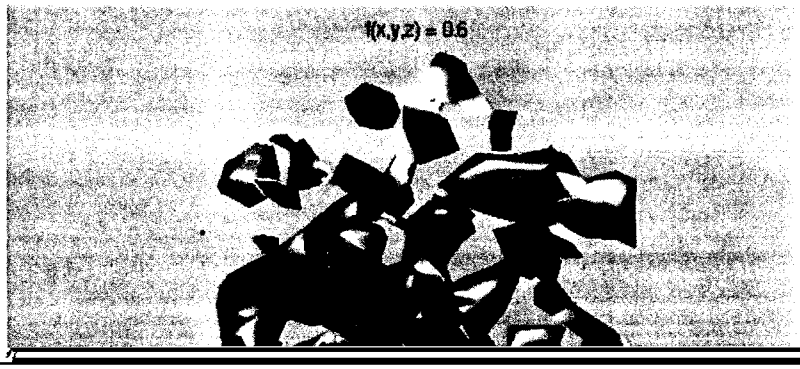
### Star Plots

Star diagrams were developed by Fienberg [1979] as a way of viewing multi-dimensional observations as a glyph or star. Each observed data point in the sample is plotted as a star, with the value of each measurement shown as a radial line from a common center point. Thus, each measured value for an observation is plotted as a spoke that is proportional to the size of the measured variable with the ends of the spokes connected with line segments to form a star. Star plots are a nice way to view the entire data set over all dimensions, but they are not suitable when there is a large number of observations ($n > 10$) or many dimensions (e.g., $d > 15$).

The next example applies this technique to data obtained from ratings of eight brands of cereal [Chakrapani and Ehrenberg, 1981; Venables and Ripley, 1994]. In our version of the star plot, the first variable is plotted as the spoke at angle $\theta = 0$, and the rest are shown counter-clockwise from there.

### Example 5.21

This example shows the MATLAB code to plot $d$-dimensional observations in a star plot. The **cereal** file contains a matrix where each row corresponds to


f(x,y,z) = 0.6

In the Computational Statistics Toolbox, we include several functions that implement some of the algorithms and graphics covered in Chapter 5. These are summarized in Table 5.3.

TABLE 5.3

List of Functions from Chapter 5 Included in the Computational Statistics Toolbox

| Purpose | MATLAB Function |
|---|---|
| Star Plot | `csstars` |
| Stem-and-leaf Plot | `csstemleaf` |
| Parallel Coordinates Plot | `csparallel` |
| Q-Q Plot | `csqqplot` |
| Poissonness Plot | `cspoissplot` |
| Andrews Curves | `csandrews` |
| Exponential Probability Plot | `csexpoplot` |
| Binomial Plot | `csbinoplot` |
| PPEDA | `csppeda`<br>`csppstrtrem`<br>`csppind` |

## 5.6 Further Reading

One of the first treatises on graphical exploratory data analysis is John Tukey's *Exploratory Data Analysis* [1977]. In this book, he explains many aspects of EDA, including smoothing techniques, graphical techniques and others. The material in this book is practical and is readily accessible to readers with rudimentary knowledge of data analysis. Another excellent book on this subject is *Graphical Exploratory Data Analysis* [du Toit, Steyn and Stumpf, 1986], which includes several techniques (e.g., Chernoff faces and profiles) that we do not cover. For texts that emphasize the visualization of technical data, see Fortner and Meyer [1997] and Fortner [1995]. The paper by Wegman, Carr and Luo [1993] discusses many of the methods we present, along with others such as stereoscopic displays, generalized nonlinear regression using skeletons and a description of $d$-dimensional grand tour. This paper and Wegman [1990] provide an excellent theoretical treatment of parallel coordinates.

*The Grammar of Graphics* by Wilkinson [1999] describes a foundation for producing graphics for scientific journals, the internet, statistical packages, or

any visualization system. It looks at the rules for producing pie charts, bar charts scatterplots, maps, function plots, and many others.

For the reader who is interested in visualization and information design, the three books by Edward Tufte are recommended. His first book, *The Visual Display of Quantitative Information* [Tufte, 1983], shows how to depict numbers. The second in the series is called *Envisioning Information* [Tufte, 1990], and illustrates how to deal with pictures of nouns (e.g., maps, aerial photographs, weather data). The third book is entitled *Visual Explanations* [Tufte, 1997], and it discusses how to illustrate pictures of verbs. These three books also provide many examples of good graphics and bad graphics. We highly recommend the book by Wainer [1997] for any statistician, engineer or data analyst. Wainer discusses the subject of good and bad graphics in a way that is accessible to the general reader.

Other techniques for visualizing multi-dimensional data have been proposed in the literature. One method introduced by Chernoff [1973] represents $d$-dimensional observations by a cartoon face, where features of the face reflect the values of the measurements. The size and shape of the nose, eyes, mouth, outline of the face and eyebrows, etc. would be determined by the value of the measurements. Chernoff faces can be used to determine simple trends in the data, but they are hard to interpret in most cases.

Another graphical EDA method that is often used is called brushing. Brushing [Venables and Ripley, 1994; Cleveland, 1993] is an interactive technique where the user can highlight data points on a scatterplot and the same points are highlighted on all other plots. For example, in a scatterplot matrix, highlighting a point in one plot shows up as highlighted in all of the others. This helps illustrate interesting structure across plots.

High-dimensional data can also be viewed using color histograms or data images. Color histograms are described in Wegman [1990]. Data images are discussed in Minotte and West [1998] and are a special case of color histograms.

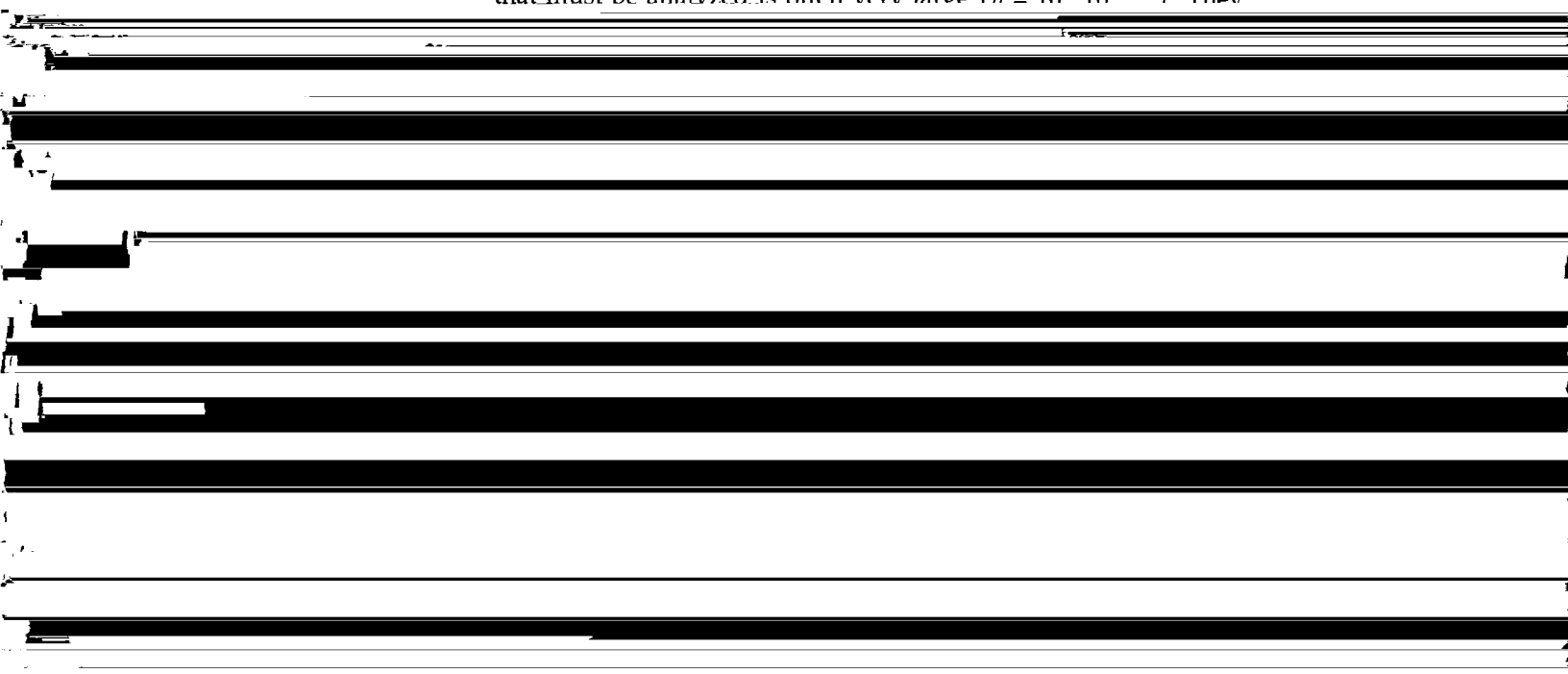
For more information on the graphical capabilities of MATLAB, we refer the reader to the MATLAB documentation *Using MATLAB Graphics*. Another excellent resource is the book called *Graphics and GUI's with MATLAB* by Marchand [1999]. These go into more detail on the graphics capabilities in MATLAB that are useful in data analysis such as lighting, use of the camera, animation, etc.

We now describe references that extend the techniques given in this book.

- ***Stem-and-leaf***: Various versions and extensions of the stem-and-leaf plot are available. We show an ordered stem-and-leaf plot in this book, but ordering is not required. Another version shades the leaves. Most introductory applied statistics books have information on stem-and-leaf plots (e.g., Montgomery, et al. [1998]). Hunter [1988] proposes an enhanced stem-and-leaf called the ***digidot plot***. This combines a stem-and-leaf with a time sequence plot. As data

are collected they are plotted as a sequence of connected dots and a stem-and-leaf is created at the same time.

- ***Discrete Quantile Plots***: Hoaglin and Tukey [1985] provide similar plots for other discrete distributions. These include the negative binomial, the geometric and the logarithmic series. They also discuss graphical techniques for plotting confidence intervals instead of points. This has the advantage of showing the confidence one has for each count.
- ***Box plots***: Other variations of the box plot have been described in the literature. See McGill, Tukey and Larsen [1978] for a discussion of the variable width box plot. With this type of display, the width of the box represents the number of observations in each sample.
- ***Scatterplots***: Scatterplot techniques are discussed in Carr, et al. [1987]. The methods presented in this paper are especially pertinent to the situation facing analysts today, where the typical data set that must be analyzed is often very large ($n = 10^3$ $10^6$ ). They



expect (with specified confidence) contains the true value of a population parameter. In hypothesis testing, our goal is to make a decision about not rejecting or rejecting some statement about the population based on data from a random sample. We give a brief summary of the concepts in classical inferential statistics, endeavoring to keep the theory to a minimum. There are many books available that contain more information on these topics. We recommend Casella and Berger [1990], Walpole and Myers [1985], Bickel and Doksum [1977], Lindgren [1993], Montgomery, Runger and Hubele [1998], and Mood, Graybill and Boes [1974].

### Hypothesis Testing

In hypothesis testing, we start with a ***statistical hypothesis***, which is a conjecture about one or more populations. Some examples of these are:

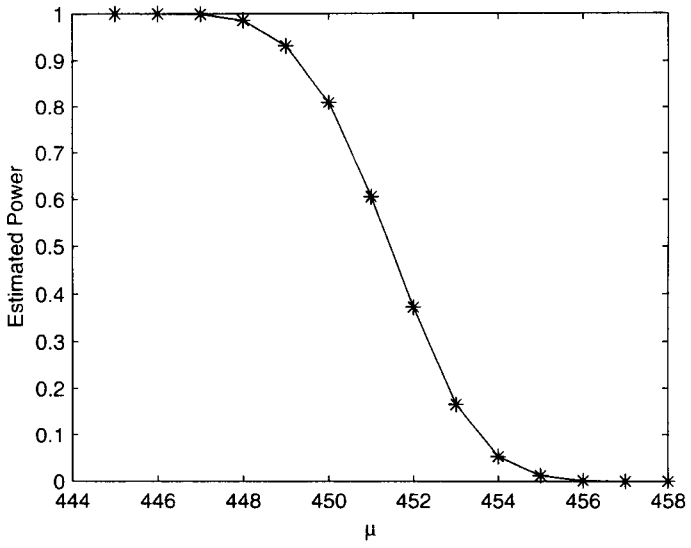



FIGURE 6.5
Here is the curve for the estimated power corresponding to the hypothesis test of Example 6.8.

An important point to keep in mind about the Monte Carlo simulations discussed in this section is that the experiment is applicable only for the situation that has been simulated. For example, when we assess the Type II error in Example 6.8, it is appropriate only for those alternative hypotheses, sample size and critical value. What would be the probability of Type II error, if some other departure from the null hypothesis is used in the simulation? In other cases, we might need to know whether the distribution of the statistic changes with sample size or skewness in the population or some other characteristic of interest. These variations are easily investigated using multiple Monte Carlo experiments.

One quantity that the researcher must determine is the number of trials that are needed in Monte Carlo simulations. This often depends on the computing assets that are available. If time and computer resources are not an issue, then $M$ should be made as large as possible. Hope [1968] showed that results from a Monte Carlo simulation are unbiased for any $M$, under the assumption that the programming is correct.

Mooney [1997] states that there is no general theory that governs the number of trials in Monte Carlo simulation. However, he recommends the following general guidelines. The researcher should first use a small number of trials and ensure that the program is working properly. Once the code has been checked, the simulation or experiments can be run for very large $M$.

Most simulations would have $M > 1000$, but $M$ between 10,000 and 25,000 is not uncommon. One important guideline for determining the number of trials, is the purpose of the simulation. If the tail of the distribution is of interest (e.g., estimating Type I error, getting $p$-values, etc.), then more trials are needed to ensure that there will be a good estimate of that area.

## 6.4 Bootstrap Methods

The treatment of the bootstrap methods described here comes from Efron and Tibshirani [1993]. The interested reader is referred to that text for more information on the underlying theory behind the bootstrap. There does not seem to be a consistent terminology in the literature for what techniques are considered bootstrap methods. Some refer to the resampling techniques of the previous section as bootstrap methods. Here, we use ***bootstrap*** to refer to Monte Carlo simulations that treat the original sample as the pseudo-population or as an estimate of the population. Thus, in the steps where we randomly sample from the pseudo-population, we now resample from the original sample.

In this section, we discuss the general bootstrap methodology, followed by some applications of the bootstrap. These include bootstrap estimates of the standard error, bootstrap estimates of bias, and bootstrap confidence intervals.

### General Bootstrap Methodology

The bootstrap is a method of Monte Carlo simulation where no parametric assumptions are made about the underlying population that generated the random sample. Instead, we use the sample as an estimate of the population. This estimate is called the empirical distribution $\hat{F}$ where each $x_i$ has probability mass $1/n$. Thus, each $x_i$ has the same likelihood of being selected in a new sample taken from $\hat{F}$.

When we use $\hat{F}$ as our pseudo-population, then we resample *with replacement* from the original sample $\mathbf{x} = (x_1, \ldots, x_n)$. We denote the new sample obtained in this manner by $\mathbf{x}^* = (x_1^*, \ldots, x_n^*)$. Since we are sampling with replacement from the original sample, there is a possibility that some points $x_i$ will appear more than once in $\mathbf{x}^*$ or maybe not at all. We are looking at the univariate situation, but the bootstrap concepts can also be applied in the $d$-dimensional case.

A small example serves to illustrate these ideas. Let's say that our random sample consists of the four numbers $\mathbf{x} = (5, 8, 3, 2)$. The following are possible samples $\mathbf{x}^*$, when we sample with replacement from $\mathbf{x}$:

$$\mathbf{x}^{*1} = (x_4, x_4, x_2, x_1) = (2, 2, 8, 5)$$
$$\mathbf{x}^{*2} = (x_4, x_2, x_3, x_4) = (2, 8, 3, 2).$$

We use the notation $\mathbf{x}^{*b}$, $b = 1, \ldots, B$ for the $b$-th bootstrap data set.

In many situations, the analyst is interested in estimating some parameter $\theta$ by calculating a statistic from the random sample. We denote this estimate by

$$\hat{\theta} = T = t(x_1, \ldots, x_n). \tag{6.11}$$

We might also like to determine the standard error in the estimate $\hat{\theta}$ and the bias. The bootstrap method can provide an estimate of this when analytical methods fail. The method is also suitable for situations when the estimator $\hat{\theta} = t(\mathbf{x})$ is complicated.

To get estimates of bias or standard error of a statistic, we obtain $B$ bootstrap samples by sampling with replacement from the original sample. For every bootstrap sample, we calculate the same statistic to obtain the ***bootstrap replications*** of $\hat{\theta}$, as follows

$$\hat{\theta}^{*b} = t(\mathbf{x}^{*b}); \qquad b = 1, \ldots, B. \tag{6.12}$$

These $B$ bootstrap replicates provide us with an estimate of the distribution of $\hat{\theta}$. This is similar to what we did in the previous section, except that we are not making any assumptions about the distribution for the original sample. Once we have the bootstrap replicates in Equation 6.12, we can use them to understand the distribution of the estimate.

The steps for the basic bootstrap methodology are given here, with detailed procedures for finding specific characteristics of $\hat{\theta}$ provided later. The issue of how large to make $B$ is addressed with each application of the bootstrap.

*PROCEDURE - BASIC BOOTSTRAP*

1. Given a random sample, $\mathbf{x} = (x_1, \ldots, x_n)$, calculate $\hat{\theta}$.
2. Sample with replacement from the original sample to get $\mathbf{x}^{*b} = (x_1^{*b}, \ldots, x_n^{*b})$.
3. Calculate the same statistic using the bootstrap sample in step 2 to get, $\hat{\theta}^{*b}$.
4. Repeat steps 2 through 3, $B$ times.
5. Use this estimate of the distribution of $\hat{\theta}$ (i.e., the bootstrap replicates) to obtain the desired characteristic (e.g., standard error, bias or confidence interval).

Efron and Tibshirani [1993] discuss a method called the ***parametric bootstrap***. In this case, the data analyst makes an assumption about the distribution that generated the original sample. Parameters for that distribution are estimated from the sample, and resampling (in step 2) is done using the assumed distribution and the estimated parameters. The parametric bootstrap is closer to the Monte Carlo methods described in the previous section.

For instance, say we have reason to believe that the data come from an exponential distribution with parameter $\lambda$. We need to estimate the variance and use

$$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{6.13}$$

as the estimator. We can use the parametric bootstrap as outlined above to understand the behavior of $\hat{\theta}$. Since we assume an exponential distribution for the data, we estimate the parameter $\lambda$ from the sample to get $\hat{\lambda}$. We then resample from an exponential distribution with parameter $\hat{\lambda}$ to get the bootstrap samples. The reader is asked to implement the parametric bootstrap in the exercises.

### Bootstrap Estimate of Standard Error

When our goal is to estimate the standard error of $\hat{\theta}$ using the bootstrap method, we proceed as outlined in the previous procedure. Once we have the estimated distribution for $\hat{\theta}$, we use it to estimate the standard error for $\hat{\theta}$. This estimate is given by

$$\hat{SE}_B(\hat{\theta}) = \left\{\frac{1}{B-1}\sum_{b=1}^{B}(\hat{\theta}^{*b} - \overline{\hat{\theta}^*})^2\right\}^{\frac{1}{2}}, \tag{6.14}$$

where

$$\overline{\hat{\theta}^*} = \frac{1}{B}\sum_{b=1}^{B}\hat{\theta}^{*b}. \tag{6.15}$$

Note that Equation 6.14 is just the sample standard deviation of the bootstrap replicates, and Equation 6.15 is the sample mean of the bootstrap replicates.

Efron and Tibshirani [1993] show that the number of bootstrap replicates $B$ should be between 50 and 200 when estimating the standard error of a statistic. Often the choice of $B$ is dictated by the computational complexity of $\hat{\theta}$, the sample size $n$, and the computer resources that are available. Even using

a small value of $B$, say $B = 25$, the analyst will gain information about the variability of $\hat{\theta}$. In most cases, taking more than 200 bootstrap replicates to estimate the standard error is unnecessary.

The procedure for finding the bootstrap estimate of the standard error is given here and is illustrated in Example 6.9

*PROCEDURE - BOOTSTRAP ESTIMATE OF THE STANDARD ERROR*

1. Given a random sample, $\mathbf{x} = (x_1, \ldots, x_n)$, calculate the statistic $\hat{\theta}$.
2. Sample with replacement from the original sample to get $\mathbf{x}^{*b} = (x_1^{*b}, \ldots, x_n^{*b})$.
3. Calculate the same statistic using the sample in step 2 to get the bootstrap replicates, $\hat{\theta}^{*b}$.
4. Repeat steps 2 through 3, $B$ times.
5. Estimate the standard error of $\hat{\theta}$ using Equations 6.14 and 6.15.

## Example 6.9

The lengths of the forearm (in inches) of 140 adult males are contained in the file **forearm** [Hand, et al., 1994]. We use these data to estimate the skewness of the population. We then estimate the standard error of this statistic using the bootstrap method. First we load the data and calculate the skewness.

```
load forearm
% Sample with replacement from this.
% First get the sample size.
n = length(forearm);
B = 100;% number of bootstrap replicates
% Get the value of the statistic of interest.
theta = skewness(forearm);
```

[redacted]

```
n = length(x);
mu = mean(x);
mr = (1/n)*sum((x-mu).^2);
```

We use this function as an input argument to **bootstrp** to get the bootstrap-$t$ confidence interval. The MATLAB code given below also shows how to get the bootstrap estimate of standard error for each bootstrap sample. First we load the data and get the observed value of the statistic.

```
load forearm
n = length(forearm);
alpha = 0.1;
B = 1000;
thetahat = mom(forearm);
```

The jackknife method is similar to cross-validation in that we leave out one observation $x_i$ from our sample to form a ***jackknife sample*** as follows

$$x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n \ .$$

This says that the $i$-th jackknife sample is the original sample with the $i$-th data point removed. We calculate the value of the estimate using this reduced jackknife sample to obtain the $i$-th ***jackknife replicate***. This is given by

$$T^{(-i)} = t(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) .$$

This means that we leave out one point at a time and use the rest of the sample to calculate our statistic. We continue to do this for the entire sample, leaving out one observation at a time, and the end result is a sequence of $n$ jackknife replications of the statistic.

The estimate of the bias of $T$ obtained from the jackknife technique is given by [Efron and Tibshirani, 1993]

$$\hat{\text{Bias}}_{jack}(T) = (n-1)(\overline{T^{(\cdot)}} - T), \tag{7.9}$$

where

$$\overline{T^{(\cdot)}} = \sum_{i=1}^{n} T^{(-i)}/n . \tag{7.10}$$

We see from Equation 7.10 that $\overline{T^{(\cdot)}}$ is simply the average of the jackknife replications of $T$.

The estimated standard error using the jackknife is defined as follows

$$\hat{SE}_{jack}(T) = \left[ \frac{n-1}{n} \sum_{i=1}^{n} (T^{(-i)} - \overline{T^{(\cdot)}})^2 \right]^{1/2} . \tag{7.11}$$

Equation 7.11 is essentially the sample standard deviation of the jackknife replications with a factor $(n-1)/n$ in front of the summation instead of $1/(n-1)$. Efron and Tibshirani [1993] show that this factor ensures that the jackknife estimate of the standard error of the sample mean, $\hat{SE}_{jack}(\bar{x})$, is an unbiased estimate.

*PROCEDURE - JACKKNIFE*

1. Leave out an observation.
2. Calculate the value of the statistic using the remaining sample points to obtain $T^{(-i)}$.
3. Repeat steps 1 and 2, leaving out one point at a time, until all $n$ $T^{(-i)}$ are recorded.
4. Calculate the jackknife estimate of the bias of $T$ using Equation 7.9.
5. Calculate the jackknife estimate of the standard error of $T$ using Equation 7.11.

The following two examples show how this is used to obtain jackknife estimates of the bias and standard error for an estimate of the correlation coefficient.

## Example 7.4

In this example, we use a data set that has been examined in Efron and Tibshirani [1993]. Note that these data are also discussed in the exercises for Chapter 6. These data consist of measurements collected on the freshman class of 82 law schools in 1973. The average score for the entering class on a national law test (`lsat`) and the average undergraduate grade point average (`gpa`) were recorded. A random sample of size $n = 15$ was taken from the population. We would like to use these sample data to estimate the correlation coefficient $\rho$ between the test scores (`lsat`) and the grade point average (`gpa`). We start off by finding the statistic of interest.

```
% Loads up a matrix - law.
load law
% Estimate the desired statistic from the sample.
lsat = law(:,1);
gpa = law(:,2);
tmp = corrcoef(gpa,lsat);
% Recall from Chapter 3 that the corrcoef function
% returns a matrix of correlation coefficients. We
% want the one in the off-diagonal position.
T = tmp(1,2);
```

We get an estimated correlation coefficient of $\hat{\rho} = 0.78$, and we would like to get an estimate of the bias and the standard error of this statistic. The following MATLAB code implements the jackknife procedure for estimating these quantities.

```
% Set up memory for jackknife replicates.
n = length(gpa);
reps = zeros(1,n);
for i = 1:n
```

```
    % Store as temporary vector:
    gpat = gpa;
    lsatt = lsat;
    % Leave i-th point out:
    gpat(i) = [];
    lsatt(i) = [];
    % Get correlation coefficient:
    % In this example, we want off-diagonal element.
    tmp = corrcoef(gpat,lsatt);
    reps(i) = tmp(1,2);
end
mureps = mean(reps);
sehat = sqrt((n-1)/n*sum((reps-mureps).^2));
% Get the estimate of the bias:
biashat = (n-1)*(mureps-T);
```

Our estimate of the standard error of the sample correlation coefficient is

$$\hat{SE}_{Jack}(\hat{\rho}) = 0.14,$$

and our estimate of the bias is

$$\hat{Bias}_{Jack}(\hat{\rho}) = -0.0065.$$

This data set will be explored further in the exercises.
□

### Example 7.5

We provide a MATLAB function called **csjack** that implements the jackknife procedure. This will work with any MATLAB function that takes the random sample as the argument and returns a statistic. This function can be one that comes with MATLAB, such as **mean** or **var**, or it can be one written by the user. We illustrate its use with a user-written function called **corr** that returns the single correlation coefficient between two univariate random variables.

```
function r = corr(data)
% This function returns the single correlation
% coefficient between two variables.
tmp = corrcoef(data);
r = tmp(1,2);
```

The data used in this example are taken from Hand, et al. [1994]. They were originally from Anscombe [1973], where they were created to illustrate the point that even though an observed value of a statistic is the same for data sets ($\hat{\rho} = 0.82$), that does not tell the entire story. He also used them to show

the importance of looking at scatterplots, because it is obvious from the plots that the relationships between the variables are not similar. The scatterplots are shown in Figure 7.3.

```
% Here is another example.
% We have 4 data sets with essentially the same
% correlation coefficient.
% The scatterplots look very different.
% When this file is loaded, you get four sets
% of x and y variables.
load anscombe
% Do the scatterplots.
subplot(2,2,1),plot(x1,y1,'k*');
subplot(2,2,2),plot(x2,y2,'k*');
subplot(2,2,3),plot(x3,y3,'k*');
subplot(2,2,4),plot(x4,y4,'k*');
```

We now determine the jackknife estimate of bias and standard error for $\hat{\theta}$

tion 7.17.

7. The lower endpoint of the confidence interval is the $\alpha_1$ quantile $\hat{q}_{\alpha_1}$ of the bootstrap replicates, and the upper endpoint of the confidence interval is the $\alpha_2$ quantile $\hat{q}_{\alpha_2}$ of the bootstrap repli-

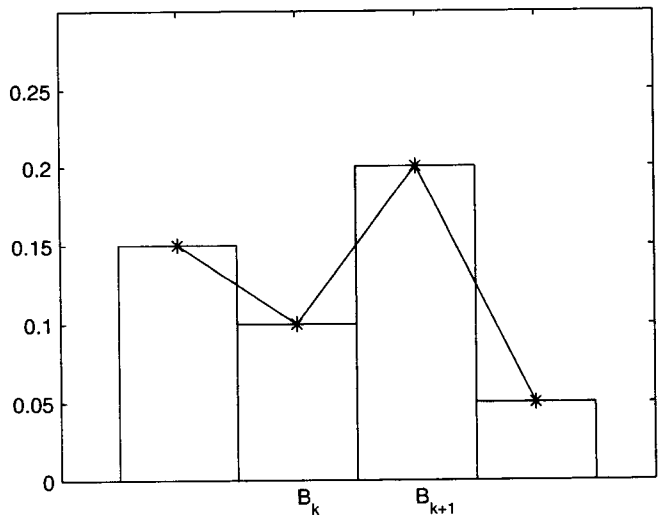



**FIGURE 8.3**
The frequency polygon is obtained by connecting the center of adjacent bins using straight lines. This figure illustrates a *section* of the frequency polygon.

*NORMAL REFERENCE RULE - FREQUENCY POLYGON*

$$h_{FP}^{*} = 2.15\sigma n^{-1/5}.$$

We can use the sample standard deviation in this rule as an estimate of $\sigma$ or choose a robust estimate based on the interquartile range. If we choose the $IQR$ and use $\hat{\sigma} = IQR/1.348$, then we obtain a bin width of

$$\hat{h}_{FP}^{*} = 1.59 \times IQR \times n^{-1/5}.$$

As for the case of histograms, Scott [1992] provides a skewness factor for frequency polygons, given by

$$\text{skewness factor}_{FP} = \frac{12^{1/5}\sigma}{e^{7\sigma^2/4}(e^{\sigma^2}-1)^{1/2}(9\sigma^4+20\sigma^2+12)^{1/5}}. \quad (8.20)$$

If there is evidence that the data come from a skewed distribution, then the bin width should be multiplied by this factor. The kurtosis factor for frequency polygons can be found in Scott [1992].

### Example 8.3

Here we show how to create a frequency polygon using the Old Faithful **geyser** data. We must first create the histogram from the data, where we use the frequency polygon Normal Reference Rule to choose the smoothing parameter.

```
load geyser
n = length(geyser);
% Use Normal Reference Rule for bin width
% of frequency polygon.
h = 2.15*sqrt(var(geyser))*n^(-1/5);
t0 = min(geyser)-1;
tm = max(geyser)+1;
bins = t0:h:tm;
vk = histc(geyser,bins);
vk(end) = [];
fhat = vk/(n*h);
```

We then use the MATLAB function called **interp1** to interpolate between the bin centers. This function takes three arguments (and an optional fourth argument). The first two arguments to **interp1** are the **xdata** and **ydata** vectors that contain the observed data. In our case, these are the bin centers and the bin heights from the density histogram. The third argument is a vector of **xinterp** values for which we would like to obtain interpolated **yinterp** values. There is an optional fourth argument that allows the user to select the type of interpolation (**linear**, **cubic**, **nearest** and **spline**). The default is **linear**, which is what we need for the frequency polygon. The following code constructs the frequency polygon for the **geyser** data.

```
% For frequency polygon, get the bin centers,
% with empty bin center on each end.
bc2 = (t0-h/2):h:(tm+h/2);
binh = [0 fhat 0];
% Use linear interpolation between bin centers
% Get the interpolated values at x.
xinterp = linspace(min(bc2),max(bc2));
fp = interp1(bc2, binh, xinterp);
```

To see how this looks, we can plot the frequency polygon and underlying histogram, which is shown in Figure 8.4.

```
% To plot this, use bar with the bin centers
tm = max(bins);
bc = (t0+h/2):h:(tm-h/2);
bar(bc,fhat,1,'w')
hold on
plot(xinterp,fp)
hold off
```

```
axis([30 120 0 0.035])
xlabel('Waiting Time (minutes)')
ylabel('Probability Density Function')
title('Old Faithful-Waiting Times Between Eruptions')
```

To ensure that we have a valid probability density function, we can verify that the area under the curve is approximately one by using the **trapz** function.

```
area = trapz(xinterp,fp);
```

We get an approximate area under the curve of 0.9998, indicating that the frequency polygon is indeed a *bona fide* density estimate.
❑

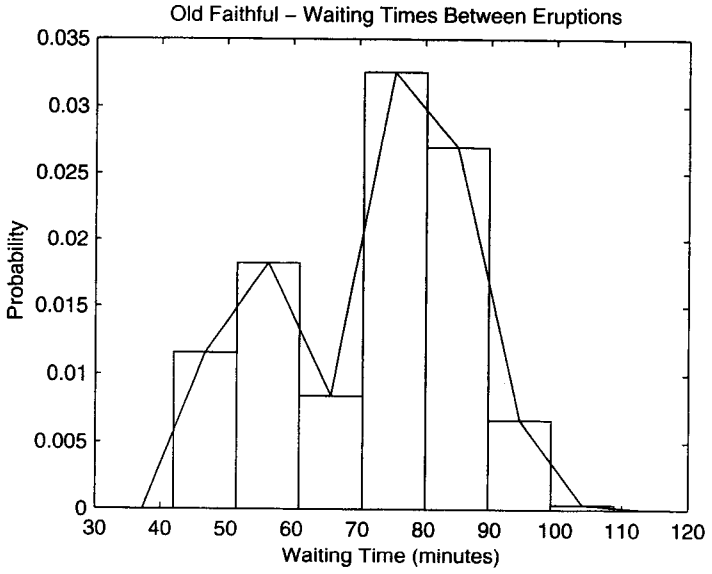



FIGURE 8.4
Frequency polygon for the Old Faithful data.

The frequency polygon can be extended to the multivariate case. The interested reader is referred to Scott [1985, 1992] for more details on the multivariate frequency polygon. He proposes an approximate Normal Reference Rule for the multivariate frequency polygon given by the following formula.

*NORMAL REFERENCE RULE - FREQUENCY POLYGON (MULTIVARIATE)*

$$h_i^* = 2\sigma_i n^{-1/(4+d)},$$

where a suitable estimate for $\sigma_i$ can be used. This is derived using the assumption that the true probability density function is multivariate normal with covariance equal to the identity matrix. The following example illustrates the procedure for obtaining a bivariate frequency polygon in MATLAB.

### Example 8.4

We first generate some random variables that are bivariate standard normal and then calculate the surface heights corresponding to the linear interpolation between the histogram density bin heights.

```
% First get the constants.
bin0 = [-4 -4];
n = 1000;
% Normal Reference Rule with sigma = 1.
h = 3*n^(-1/4)*ones(1,2);
% Generate bivariate standard normal variables.
x = randn(n,2);
% Find the number of bins.
nb1 = ceil((max(x(:,1))-bin0(1))/h(1));
nb2 = ceil((max(x(:,2))-bin0(2))/h(2));
% Find the mesh or bin edges.
t1 = bin0(1):h(1):(nb1*h(1)+bin0(1));
t2 = bin0(2):h(2):(nb2*h(2)+bin0(2));
[X,Y] = meshgrid(t1,t2);
```

Now that we have the random variables and the bin edges, the next step is to find the number of observations that fall into each bin. This is easily done with the MATLAB function **inpolygon**. This function can be used with any polygon (e.g., triangle or hexagon), and it returns the indices to the points that fall into that polygon.

```
% Find bin frequencies.
[nr,nc] = size(X);
vu = zeros(nr-1,nc-1);
for i = 1:(nr-1)
 for j = 1:(nc-1)
   xv = [X(i,j) X(i,j+1) X(i+1,j+1) X(i+1,j)];
   yv = [Y(i,j) Y(i,j+1) Y(i+1,j+1) Y(i+1,j)];
   in = inpolygon(x(:,1),x(:,2),xv,yv);
   vu(i,j) = sum(in(:));
 end
end
```

```
fhat = vu/(n*h(1)*h(2));
```

Now that we have the histogram density, we can use the MATLAB function **interp2** to linearly interpolate at points between the bin centers.

```
% Now get the bin centers for the frequency polygon.
% We add bins at the edges with zero height.
t1 = (bin0(1)-h(1)/2):h(1):(max(t1)+h(1)/2);
t2 = (bin0(2)-h(2)/2):h(2):(max(t2)+h(2)/2);
[bcx,bcy] = meshgrid(t1,t2);
[nr,nc] = size(fhat);
binh = zeros(nr+2,nc+2);  % add zero bin heights
binh(2:(1+nr),2:(1+nc))=fhat;
% Get points where we want to interpolate to get
% the frequency polygon.
[xint,yint]=meshgrid(linspace(min(t1),max(t1),30)
```

properties of the kernel function, such as continuity and differentiability..

From Equation 8.26, the estimated probability density function is obtained by placing a weighted kernel function, centered at each data point and then taking the average of them. See Figure 8.8 for an illustration of this procedure.

$$f(x) = 0.3 \times \phi(x;-3, 1) + 0.3 \times \phi(x;0, 1) + 0.4 \times \phi(x;2, 0.5).$$

```
% Get the true model to generate data.
pi_tru = [0.3 0.3 0.4];
n = 100;
x = zeros(n,1);
% Now generate 100 random variables. First find
% the number that fall in each one.
r = rand(1,100);
% Find the number generated from each component.
ind1 = length(find(r <= 0.3));
ind2 = length(find(r > 0.3 & r <= 0.6));
ind3 = length(find(r > 0.6));
% create some artificial  3 term mixture data
x(1:ind1) = randn(ind1,1) - 3;
x(ind1+1:ind2+ind1)=randn(ind2,1);
x(ind1+ind2+1:n) = randn(ind3,1)*sqrt(0.5)+2;
```

We now call the adaptive mixtures function **csadpmix** to estimate the model.

```
% Now call the adaptive mixtures function.
maxterms = 25;
[pihat,muhat,varhat] = csadpmix(x,maxterms);
```

The following MATLAB commands provide the plots shown in Figure 8.16.

```
% Get the plots.
csdfplot(muhat,varhat,pihat,min(x),max(x));
axis equal
nterms = length(pihat);
figure
csplotuni(pihat,muhat,varhat,...
    nterms,min(x)-5,max(x)+5,100)
```

We reorder the observations and repeat the process to get the plots in Figure 8.17.

```
% Now re-order the points and repeat
% the adaptive mixtures process.
ind = randperm(n);
x = x(ind);
[pihat,muhat,varhat] = csadpmix(x,maxterms);
```

❑

Our example above demonstrates some interesting things to consider with adaptive mixtures. First, the model complexity or the number of terms is sometimes greater than is needed. For example, in Figure 8.16, we show a *dF*

plot for the three term mixture model in Example 8.12. Note that the adaptive mixture approach yields more than three terms. This is a problem with mixture models in general. Different models (i.e., number of terms and estimated component parameters) can produce essentially the same function estimate or curve for $\hat{f}(\mathbf{x})$. This is illustrated in Figures 8.16 and 8.17, where we see that similar curves are obtained from two different models for the same data set. These results are straight from the adaptive mixtures density estimation approach. In other words, we did not use this estimate as an initial starting point for the EM approach. If we had applied the iterative EM to these estimated models, then the curves should be the same.

The other issue that must be considered when using the adaptive mixtures approach is that the resulting model or estimated probability density function depends on the order in which the data are presented to the algorithm. This is also illustrated in Figures 8.16 and 8.17, where the second estimated model is obtained after re-ordering the data. These issues were addressed by Solka [1995].

## 8.5 Generating Random Variables

In the introduction, we discussed several uses of probability density estimates, and it is our hope that the reader will discover many more. One of the applications of density estimation is in the area of modeling and simulation. Recall that a key aspect of modeling and simulation is the collection of data generated according to some underlying random process and the desire to generate more random variables from the same process for simulation purposes. One option is to use one of the density estimation techniques discussed in this chapter and randomly sample from that distribution. In this section, we provide the methodology for generating random variables from finite or adaptive mixtures density estimates.

We have already seen an example of this procedure in Example 8.11 and Example 8.12. The procedure is to first choose the class membership of generated observations based on uniform (0,1) random variables. The number of random variables generated from each component density is given by the corresponding proportion of these uniform variables that are in the required range. The steps are outlined here.

*PROCEDURE - GENERATING RANDOM VARIABLES (FINITE MIXTURE)*

1. We are given a finite mixture model $(p_i, g_i(\mathbf{x};\theta_i))$ with $c$ components, and we want to generate $n$ random variables from that distribution.

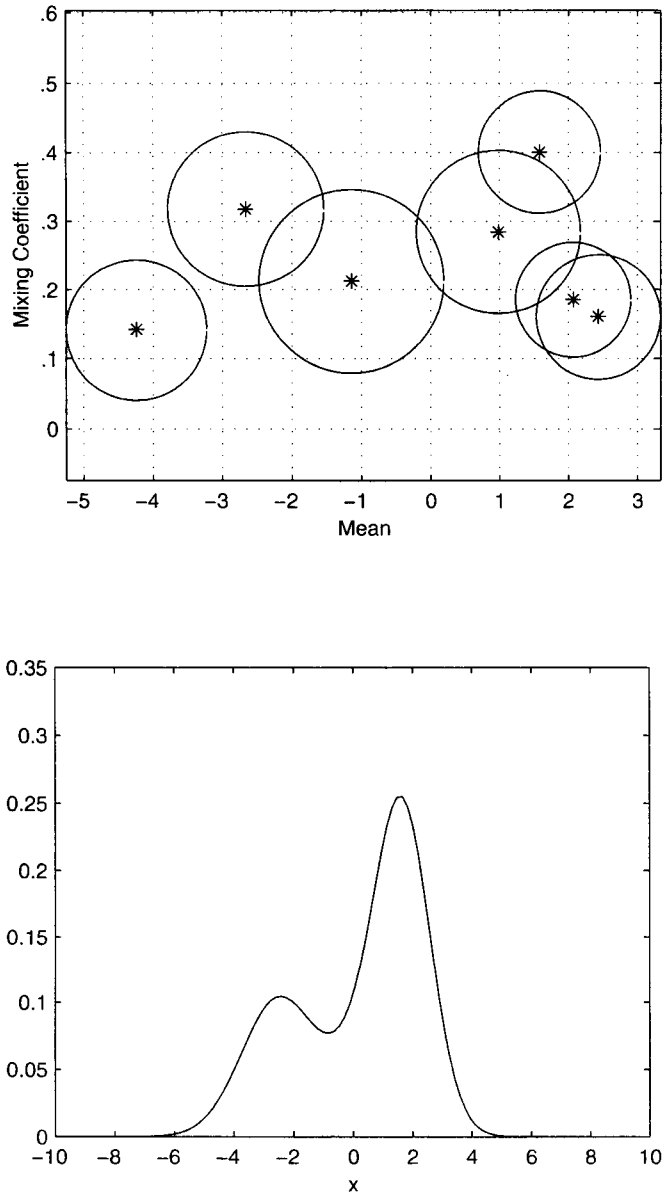



FIGURE 8.16
The upper plot shows the $dF$ representation for Example 8.12. Compare this with Figure 8.17 for the same data. Note that the curves are essentially the same, but the number of terms and associated parameters are different. Thus, we can get different models for the same data.

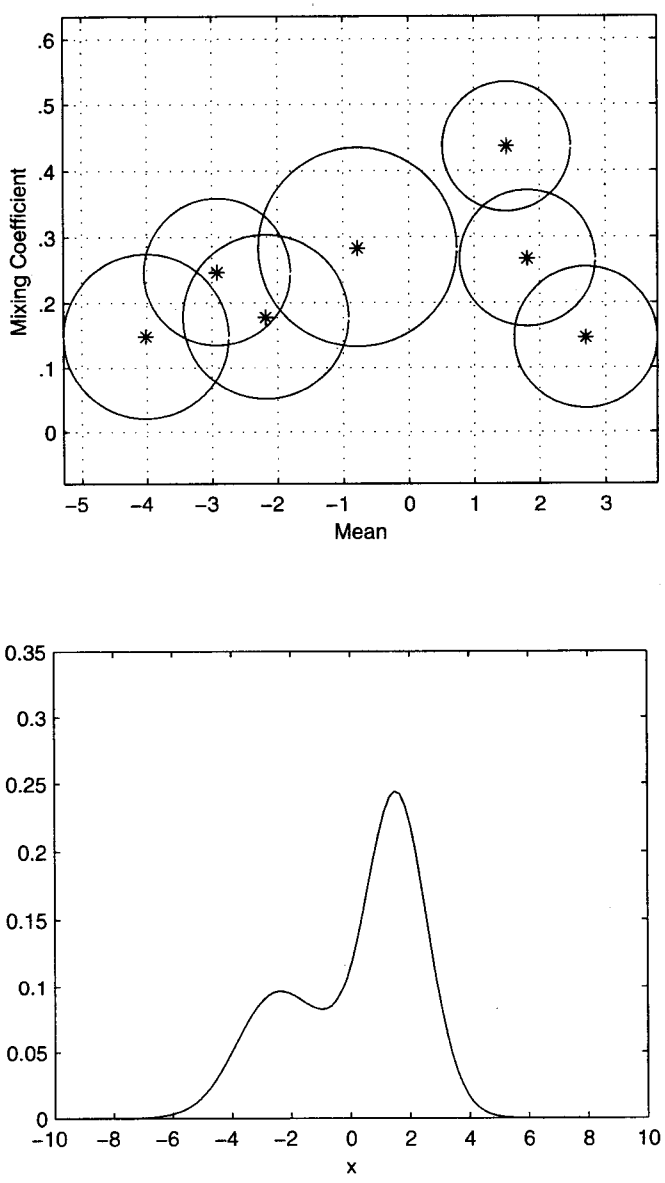



FIGURE 8.17
This is the second estimated model using adaptive mixtures for the data generated in Example 8.12. This second model was obtained by re-ordering the data set and then implementing the adaptive mixtures technique. This shows the dependence of the technique on the order in which the data are presented to the method.

2. First determine the component membership of each of the $n$ random variables. We do this by generating $n$ uniform (0,1) random variables ($U_i$). Component membership is determined as follows

   If $0 \le U_i < p_1$, then $X_i$ is from component density 1.

   If $p_1 \le U_i < p_1 + p_2$, then $X_i$ is from component density 2.

   . . .

   If $\sum_{j=1}^{c-1} p_j \le U_i \le 1$, then $X_i$ is from component density $c$.

3. Generate the $X_i$ from the corresponding $g_i(\mathbf{x};\theta_i)$ using the compo-

you want to get the value of the estimate, evaluate a weighted kernel at each point, and take the average. The following code shows you how to do this. Implement this using the Buffalo **snowfall** data. Verify that this is a valid density by estimating the area under the curve.

```
load snowfall
x = 0:140;
n = length(snowfall);
h = 1.06*sqrt(var(snowfall))*n^(-1/5);
fhat = zeros(size(x));
% Loop over all values of x in the domain
% to get the kernel evaluated at that point.
for i = 1:length(x)
 xloc = x(i)*ones(1,n);
 % Take each value of x and evaluate it at
 % n weighted kernels -
 % each one centered at a data point, then add them up.
 arg = ((xloc-snowfall)/h).^2;
 fhat(i) = (sum(exp(-.5*(arg)))/(n*h*sqrt(2*pi)));
end
```

8.12. Write a MATLAB function that will construct a kernel density esti-

```
% based on the class-conditional probabilities.
ncc = 0;
% We will use only the first two features of
% the iris data for our classification.
% This should make it more difficult to
% separate the classes.
% Delete 3rd and 4th features.
virginica(:,3:4) = [];
versicolor(:,3:4) = [];
[nver,d] = size(versicolor);
[nvir,d] = size(virginica);
n = nvir + nver;
```

First, we will loop through all of the **versicolor** observations. We build a classifier, leaving out one pattern at a time for testing purposes. Throughout this loop, the class-conditional probability for **virginica** remains the same, so we find that first.

```
% Loop first through all of the patterns corresponding
% to versicolor. Here correct classification
% is obtained if pxgver > pxgvir;
muvir = mean(virginica);
covvir = cov(virginica);
% These will be the same for this part.
for i = 1:nver
   % Get the test point and the training set
   versitrain = versicolor;
   % This is the testing point.
   x = versitrain(i,:);
   % Delete from training set.
   % The result is the training set.
   versitrain(i,:)=[];
   muver = mean(versitrain);
   covver = cov(versitrain);
   pxgver = csevalnorm(x,muver,covver);
   pxgvir = csevalnorm(x,muvir,covvir);
   if pxgver > pxgvir
   % then we correctly classified it
     ncc = ncc+1;
   end
end
```

We repeat the same procedure leaving out each **virginica** observation as the test pattern.

```
% Loop through all of the patterns of virginica notes.
% Here correct classification is obtained when
% pxgvir > pxxgver
```

```
muver = mean(versicolor);
covver = cov(versicolor);
% Those remain the same for the following.
for i = 1:nvir
   % Get the test point and training set.
   virtrain = virginica;
   x = virtrain(i,:);
   virtrain(i,:)=[];
   muvir = mean(virtrain);
   covvir = cov(virtrain);
   pxgver = csevalnorm(x,muver,covver);
   pxgvir = csevalnorm(x,muvir,covvir);
   if pxgvir > pxgver
     % then we correctly classified it
     ncc = ncc+1;
   end
end
```

Finally, the probability of correct classification is estimated using

```
pcc = ncc/n;
```

The estimated probability of correct classification for the **iris** data using cross-validation is 0.68.

❑

### Receiver Operating Characteristic (ROC) Curve

We now turn our attention to how we can use cross-validation to evaluate a classifier that uses the likelihood approach with varying decision thresholds $\tau_C$. It would be useful to understand how the classifier performs for various thresholds (corresponding to the probability of false alarm) of the likelihood ratio. This will tell us what performance degradation we have (in terms of correctly classifying the target class) if we limit the probability of false alarm to some level.

We start by dividing the sample into two sets: one with all of the target observations and one with the non-target patterns. Denote the observations as follows

$$\mathbf{x}_i^{(1)} \Rightarrow \text{Target pattern } (\omega_1)$$
$$\mathbf{x}_i^{(2)} \Rightarrow \text{Non-target pattern } (\omega_2).$$

Let $n_1$ represent the number of target observations (class $\omega_1$) and $n_2$ denote the number of non-target (class $\omega_2$) patterns. We work first with the non-target observations to determine the threshold we need to get a desired proba-

bility of false alarm. Once we have the threshold, we can determine the probability of correctly classifying the observations belonging to the target class.

Before we go on to describe the receiver operating characteristic (ROC) curve, we first describe some terminology. For any boundary we might set for the decision regions, we are likely to make mistakes in classifying cases. There will be some target patterns that we correctly classify as targets and some we misclassify as non-targets. Similarly, there will be non-target patterns that are correctly classified as non-targets and some that are misclassified as targets. This is summarized as follows:

- True Positives - TP: This is the fraction of patterns correctly classified as target cases.
- False Positives - FP: This is the fraction of non-target patterns incorrectly classified as target cases.
- True Negatives - TN: This is the fraction of non-target cases correctly classified as non-target.
- False Negatives - FN: This is the fraction of target cases incorrectly classified as non-target.

In our previous terminology, the false positives (FP) correspond to the false alarms. Figure 9.8 shows these areas for a given decision boundary.

A **ROC *curve*** is a plot of the true positive rate against the false positive rate. ROC curves are used primarily in signal detection and medical diagnosis [Egan, 1975; Lusted, 1971; McNeil, et. al., 1975; Hanley and McNeil, 1983; Hanley and Hajian-Tilaki, 1997]. In their terminology, the true positive rate is also called the sensitivity. ***Sensitivity*** is the probability that a classifier will classify a pattern as a target when it really is a target. ***Specificity*** is the probability that a classifier will correctly classify the true non-target cases. Therefore, we see that a ROC curve is also a plot of sensitivity against 1 minus specificity.

One of the purposes of a ROC curve is to measure the discriminating power of the classifier. It is used in the medical community to evaluate the diagnostic power of tests for diseases. By looking at a ROC curve, we can understand the following about a classifier:

- It shows the trade-off between the probability of correctly classifying the target class (sensitivity) and the false alarm rate (1 – specificity).
- The area under the ROC curve can be used to compare the performance of classifiers.

We now show in more detail how to construct a ROC curve. Recall that the likelihood ratio is given by

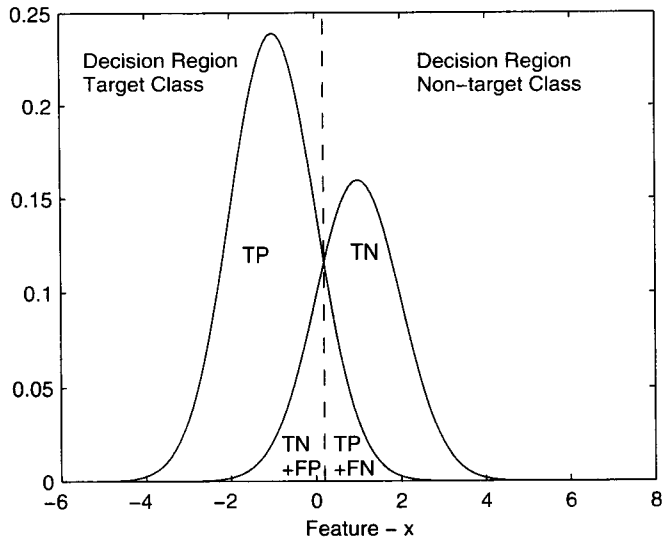



**FIGURE 9.8**
In this figure, we see the decision regions for deciding whether a feature corresponds to the target class or the non-target class.

$$L_R(\mathbf{x}) = \frac{P(\mathbf{x}|\omega_1)}{P(\mathbf{x}|\omega_2)}.$$

We start off by forming the likelihood ratios using the non-target ($\omega_2$) observations and cross-validation to get the distribution of the likelihood ratios when the class membership is truly $\omega_2$. We use these likelihood ratios to set

$t_L$ and $t_R$ are the left and right child nodes.

$\{t_1\}$ is the tree containing only the root node.

$T_t$ is a branch of tree $T$ starting at node $t$.

$\widetilde{T}$ is the set of terminal nodes in the tree.

$|\widetilde{T}|$ is the number of terminal nodes in tree $T$.

$t_k^*$ is the node that is the weakest link in tree $T_k$.

$n$ is the total number of observations in the learning set.

$n_j$ is the number of observations in the learning set that belong to the $j$-th class $\omega_j$, $j = 1, \ldots, J$.

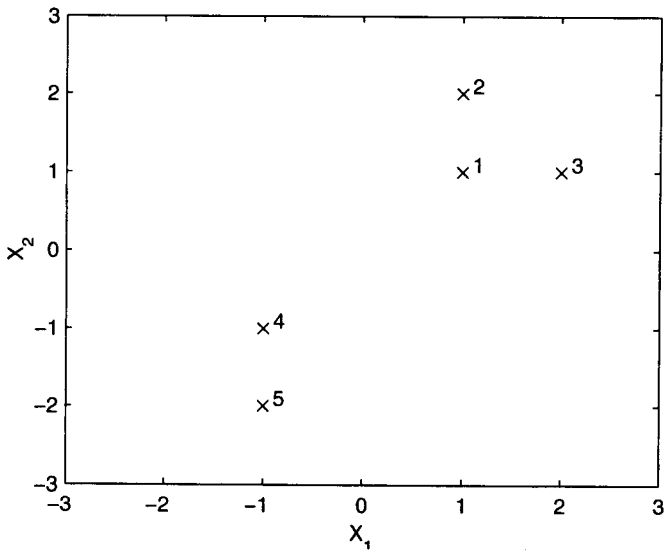



FIGURE 9.16
These are the observations used in Example 9.15. Two clusters are clearly seen.

centroid linkage and Ward's method. The MATLAB Statistics Toolbox provides a function called `linkage` that will perform agglomerative clustering using any of these methods. Its use is illustrated in the next example, but first we briefly describe each of the methods [Hair, et al., 1995].

The ***single linkage*** method uses minimum distance, where the distance between clusters is defined as the distance between the closest pair of observations. Pairs consisting of one case from each group are used in the calculation. The first cluster is formed by merging the two groups with the shortest distance. Then the next smallest distance is found between all of the clusters (keep in mind that an observation is also a cluster). The two clusters corresponding to the smallest distance are then merged. The process continues in this manner until there is one group. In some cases, single linkage can lead to chaining of the observations, where those on the ends of the chain might be very dissimilar.

The process for the ***complete linkage*** method is similar to single linkage, but the clustering criterion is different. The distance between groups is defined as the most distant pair of observations, with one coming from each group. The logic behind using this type of similarity criterion is that the maximum distance between observations in each cluster represents the smallest sphere that can enclose all of the objects in both clusters. Thus, the closest of these cluster pairs should be grouped together. The complete linkage method does not have the chaining problem that single linkage has.

The ***average linkage*** method for clustering starts out the same way as single and complete linkage. In this case, the cluster criterion is the average distance between all pairs, where one member of the pair comes from each cluster. Thus, we find all pairwise distances between observations in each cluster and take the average. This linkage method tends to combine clusters with small variances and to produce clusters with approximately equal variance.

***Centroid linkage*** calculates the distance between two clusters as the distance between the centroids. The ***centroid*** of a cluster is defined as the $d$-dimensional sample mean for those observations that belong to the cluster. Whenever we merge clusters together or add an observation to a cluster, the centroid is recalculated.

The distance between two clusters using ***Ward's linkage*** method is defined as the incremental sum of the squares between two clusters. To merge clusters, the within-group sum-of-squares is minimized over all possible partitions obtained by combining two clusters. The within-group sum-of-squares is defined as the sum of the squared distances between all observations in a cluster and its centroid. This method tends to produce clusters with approximately the same number of observations in each one.

### Example 9.16

We illustrate the `linkage` function using the data and distances from the previous example. We look only at single linkage and complete linkage using the Euclidean distances. We show the results of the clustering in dendrograms given in Figures 9.17 and 9.18.

```
% Get the cluster output from the linkage function.
zsingle = linkage(ye,'single');
zcomplete = linkage(ye,'complete');
% Get the dendrogram.
dendrogram(zsingle)
title('Clustering - Single Linkage')
dendrogram(zcomplete)
title('Clustering - Complete Linkage')
```

A ***dendrogram*** shows the links between objects as inverted U-shaped lines, where the height of the U represents the distance between the objects. The cases are listed along the horizontal axis. Cutting the tree at various $y$ values of the dendrogram yields different clusters. For example, cutting the complete linkage tree at $y = 1.2$ would yield 3 clusters. As expected, if we choose to create two clusters, then the two linkage methods give the same cluster definitions.

❑

Now that we have our cases clustered, we would like to measure the validity of the clustering. One way to do this would be to compare the distances between all observations with the links in the dendrogram. If the clustering

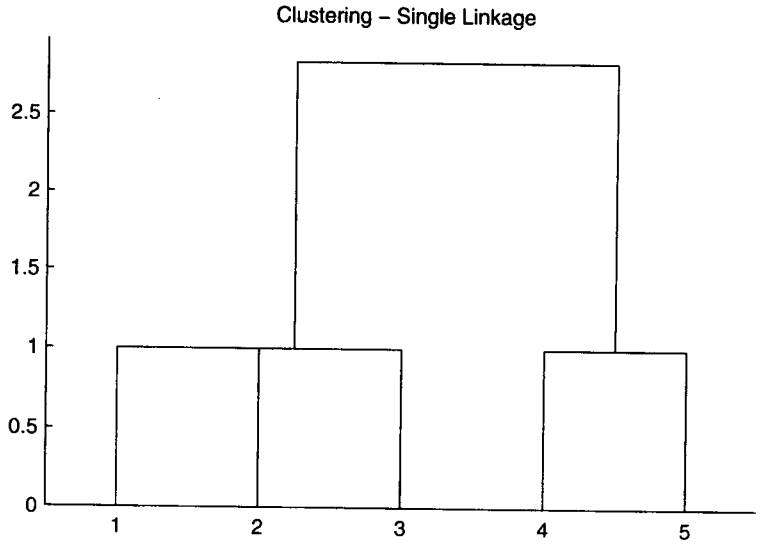



**FIGURE 9.17**
This is the dendrogram using Euclidean distances and single linkage.

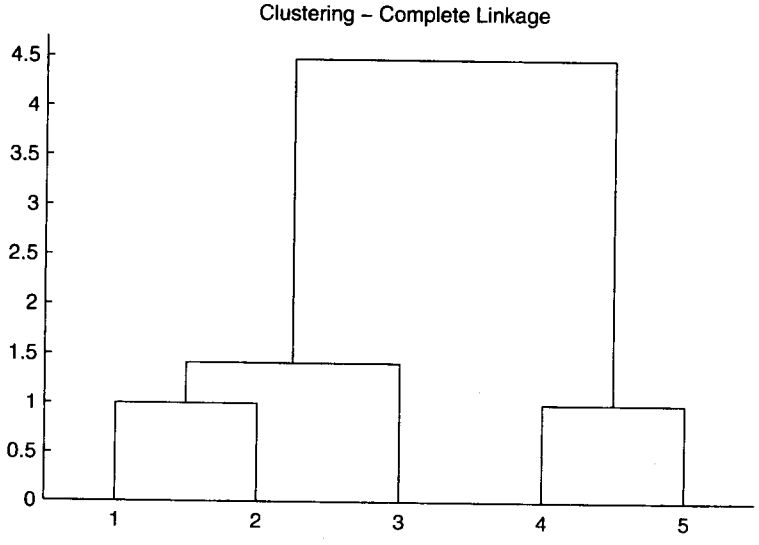



**FIGURE 9.18**
This is the dendrogram using Euclidean distances and complete linkage.

is a valid one, then there should be a strong correlation between them. We can measure this using the ***cophenetic correlation coefficient***. A cophenetic matrix is defined using the results of the linkage procedure. The $ij$-th entry of the cophenetic matrix is the fusion level at which the $i$-th and $j$-th objects appear together in the same cluster for the first time. The correlation coefficient between the distances and the corresponding cophenetic entries is the cophenetic correlation coefficient. Large values indicate that the linkage provides a reasonable clustering of the data. The MATLAB Statistics Toolbox provides a function that will calculate the cophenetic correlation coefficient. Its use is illustrated in the following example.

### Example 9.17

In this example, we show how to obtain the cophenetic correlation coefficient in MATLAB. We use the same small data set from before and calculate the cophenetic correlation coefficient when we have clusters based on different distances and linkages. First, we get the clusters using the following commands.

```
x = [1 1; 1 2; 2 1; -1 -1; -1 -2];
ye = pdist(x,'euclid');
ycb = pdist(x,'cityblock');
zsineu = linkage(ye,'single');
zcompeu = linkage(ye,'complete');
zsincb = linkage(ycb,'single');
zcomcb = linkage(ycb,'complete');
```

We now have four different cluster hierarchies. Their cophenetic correlation coefficients can be found from the following:

```
ccompeu = cophenet(zcompeu,ye);
csineu = cophenet(zsineu,ye);
csincb = cophenet(zsincb,ycb);
ccomcb = cophenet(zcomcb,ycb);
```

As expected, all of the resulting cophenetic correlation coefficients are large (approximately 0.95), with the largest corresponding to the complete linkage clustering based on the city block distance.
❑

#### *K*-Means Clustering

The goal of $k$-means clustering is to partition the data into $k$ groups such that the within-group sum-of-squares is minimized. One way this technique differs from hierarchical clustering is that we must specify the number of groups or clusters that we are looking for. We briefly describe two algorithms for obtaining clusters via $k$-means.

One of the basic algorithms for $k$-means clustering is a two step procedure. First, we assign observations to its closest group, usually using the Euclidean distance between the observation and the cluster centroid. The second step of the procedure is to calculate the new cluster centroid using the assigned objects. These steps are alternated until there are no changes in cluster membership or until the centroids do not change. This algorithm is sometimes referred to as HMEANS [Spath, 1980] or the basic ISODATA method.

*PROCEDURE - HMEANS ALGORITHM*

1. Specify the number of clusters $k$.
2. Determine initial cluster centroids. These can be randomly chosen or the user can specify them.
3. Calculate the distance between each observation and each cluster centroid.
4. Assign every observation to the closest cluster.
5. Calculate the centroid (i.e., the $d$-dimensional mean) of every cluster using the observations that were just grouped there.
6. Repeat steps 3 through 5 until no more changes are made.

There are two problems with the HMEANS algorithm. The first one is that this method could lead to empty clusters, so users should be aware of this possibility. As the centroid is recalculated and observations are reassigned to groups, some clusters could become empty. The second issue concerns the optimality of the partitions. With $k$-means, we are searching for partitions where the within-group sum-of-squares is minimum. It can be shown [Webb, 1999] that in some cases, the final $k$-means cluster assignment is not optimal, in the sense that moving a single point from one cluster to another may reduce the sum of squared errors. The following procedure helps address the second problem.

*PROCEDURE - K-MEANS*

1. Obtain a partition of $k$ groups, possibly from the HMEANS algorithm.
2. Take each data point $\mathbf{x}_i$ and calculate the Euclidean distance between it and every cluster centroid.
3. Here $\mathbf{x}_i$ is in the $r$-th cluster, $n_r$ is the number of points in the $r$-th cluster, and $d_{ir}^2$ is the Euclidean distance between $\mathbf{x}_i$ and the centroid of cluster $r$. If there is a group $s$ such that

$$\frac{n_r}{n_r - 1} d_{ir}^2 > \frac{n_s}{n_s + 1} d_{is}^2,$$

then move $\mathbf{x}_i$ to cluster $s$.

4. If there are several clusters that satisfy the above inequality, then move the $\mathbf{x}_i$ to the group that has the smallest value for

$$\frac{n_s}{n_s + 1} d_{is}^2 .$$

5. Repeat steps 2 through 4 until no more changes are made.

We note that there are many algorithms for $k$-means clustering described in the literature. We provide some references to these in the last section of this chapter.

### Example 9.18

We show how to implement HMEANS in MATLAB, using the `iris` data. Normally, clustering methods would be used on data where we do not know what groups are there, unlike the `iris` data. However, since we do know the true groups represented by the data, these will give us a way to verify that the clusters make sense. We first obtain the cluster centers by randomly picking observations from the data set. Note that initial cluster centers do not have to be actual observations.

```
load iris
k = 3;
% Put all of the data together.
x = [setosa;versicolor;virginica];
[n,d] = size(x);

% Pick some observations to be the cluster centers.
ind = randperm(n);
ind = ind(1:k);
nc = x(ind,:);
% Set up storage.
% Integers 1,...,k indicating cluster membership
cid = zeros(1,n);
% Make this different to get the loop started.
oldcid = ones(1,n);
% The number in each cluster.
nr = zeros(1,k);
% Set up maximum number of iterations.
maxiter = 100;
iter = 1;

while ~isequal(cid,oldcid)& iter < maxiter
    oldcid = cid;
```

```
        % Implement the hmeans algorithm.
        % For each point, find the distance
        % to all cluster centers.
        for i = 1:n
            dist = sum((repmat(x(i,:),k,1)-nc).^2,2);
             % assign it to this cluster
            [m,ind] = min(dist);
            cid(i) = ind;
        end
        % Find the new cluster centers.
        for i = 1:k
            % Find all points in this cluster.
            ind = find(cid==i);
            % Find the centroid.
            nc(i,:) = mean(x(ind,:));
            % Find the number in each cluster;
            nr(i) = length(ind);
        end
        iter = iter + 1
    end
```

To check these results, we show a scatterplot of the *first two features* of the `iris` data in Figure 9.19, where the three classes are represented by different plotting symbols. The clusters we obtain from this implementation of $k$-means clustering (using the HMEANS procedure) are shown in Figure 9.20. The algorithm finds the one group, corresponding to *Iris setosa*, but has trouble separating the other two species. However, the results are certainly reasonable.

□

## 9.6 MATLAB Code

We provide a function called **cshmeans** that implements the HMEANS algorithm given above. We also have a function called **cskmeans** that checks to see if moving individual observations changes the sum-square error. With both of these functions, the user can specify the initial centers as an input argument. However, if that argument is omitted, then the function will randomly pick the initial cluster centers.

As we stated in the body of the text, there are many MATLAB functions available that the analyst can use to develop classifiers using Bayes decision theory. These are any of the functions in the Statistics Toolbox that estimates a probability density function using the parametric approach: **normfit**, **gamfit**, **weibfit**, ...

approach, estimating the class-conditional probabilities using the product kernel. Which classifier performs better, based on the ROC curve analysis?

9.9 For the [illegible] data, obtain a classification tree. Use the pruning

We now give the expression for the local polynomial kernel estimator. Let $d$ represent the degree of the polynomial that we fit at a point $x$. We obtain the estimate $\hat{y} = \hat{f}(x)$ by fitting the polynomial

$$\beta_0 + \beta_1(X_i - x) + \ldots + \beta_d(X_i - x)^d \tag{10.15}$$

using the points $(X_i, Y_i)$ and utilizing the weighted least squares procedure. The weights are given by the kernel function

$$K_h(X_i - x) = \frac{1}{h}K\left(\frac{X_i - x}{h}\right). \tag{10.16}$$

The value of the estimate at a point $x$ is $\hat{\beta}_0$, where the $\hat{\beta}_i$ minimize

$$\sum_{i=1}^{n} K_h(X_i - x)(Y_i - \beta_0 - \beta_1(X_i - x) - \ldots - \beta_d(X_i - x)^d)^2. \tag{10.17}$$

Because the points that are used to estimate the model are all centered at $x$ (see Equation 10.15), the estimate at $x$ is obtained by setting the argument in the model equal to zero. Thus, the only parameter left is the constant term $\hat{\beta}_0$.

The attentive reader will note that the argument of the $K_h$ is backwards from what we had in probability density estimation using kernels. There, the kernels were centered at the random variables $X_i$. We follow the notation of Wand and Jones [1995] that shows explicitly that we are centering the kernels at the points $x$ where we want to obtain the estimated value of the function.

We can write this weighted least squares procedure using matrix notation. According to standard weighted least squares theory [Draper and Smith, 1981], the solution can be written as

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}_x^T \mathbf{W}_x \mathbf{X}_x)^{-1} \mathbf{X}_x^T \mathbf{W}_x \mathbf{Y}, \tag{10.18}$$

where $\mathbf{Y}$ is the $n \times 1$ vector of responses,

$$\mathbf{X}_x = \begin{bmatrix} 1 & X_1 - x & \ldots & (X_1 - x)^d \\ \vdots & \vdots & \ldots & \vdots \\ 1 & X_n - x & \ldots & (X_n - x)^d \end{bmatrix}, \tag{10.19}$$

and $\mathbf{W}_x$ is an $n \times n$ matrix with the weights along the diagonal. These weights are given by

$$w_{ii}(x) = K_h(X_i - x). \tag{10.20}$$

Some of these weights might be zero depending on the kernel that is used. The estimator $\hat{y} = \hat{f}(x)$ is the intercept coefficient $\beta_0$ of the local fit, so we can obtain the value from

$$\hat{f}(x) = \mathbf{e}_1^T(\mathbf{X}_x^T\mathbf{W}_x\mathbf{X}_x)^{-1}\mathbf{X}_x^T\mathbf{W}_x\mathbf{Y} \tag{10.21}$$

where $\mathbf{e}_1^T$ is a vector of dimension $(d+1)\times 1$ with a one in the first place and zeroes everywhere else.

### Nadaraya-Watson Estimator

Some explicit expressions exist when $d = 0$ and $d = 1$. When $d$ is zero, we fit a constant function locally at a given point $x$. This estimator was developed separately by Nadaraya [1964] and Watson [1964]. The Nadaraya-Watson estimator is given below.

*NADARAYA-WATSON KERNEL ESTIMATOR:*

$$\hat{f}_{NW}(x) = \frac{\sum_{i=1}^{n} K_h(X_i - x)Y_i}{\sum_{i=1}^{n} K_h(X_i - x)}. \tag{10.22}$$

Note that this is for the case of a random design. When the design points are fixed, then the $X_i$ is replaced by $x_i$, but otherwise the expression is the same [Wand and Jones, 1995].

There is an alternative estimator that can be used in the *fixed design* case. This is called the Priestley-Chao kernel estimator [Simonoff, 1996].

*PRIESTLEY-CHAO KERNEL ESTIMATOR:*

$$\hat{f}_{PC}(x) = \frac{1}{h}\sum_{i=1}^{n}(x_i - x_{i-1})K\left(\frac{x - x_i}{h}\right)y_i, \tag{10.23}$$

where the $x_i$, $i = 1, \ldots, n$, represent a fixed set of ordered nonrandom numbers. The Nadarya-Watson estimator is illustrated in Example 10.6, while the Priestley-Chao estimator is saved for the exercises.

### Example 10.6

We show how to implement the Nadarya-Watson estimator in MATLAB. As in the previous example, we generate data that follows a sine wave with added noise.

```
% Generate some noisy data.
x = linspace(0, 4 * pi,100);
y = sin(x) + 0.75*randn(size(x));
```

The next step is to create a MATLAB **inline** function so we can evaluate the weights. Note that we are using the normal kernel.

```
% Create an inline function to evaluate the weights.
mystrg='(2*pi*h^2)^(-1/2)*exp(-0.5*((x - mu)/h).^2)';
wfun = inline(mystrg);
```

We now get the estimates at each value of $x$.

```
% Set up the space to store the estimated values.
% We will get the estimate at all values of x.
yhatnw = zeros(size(x));
n = length(x);
% Set the window width.
h = 1;
% find smooth at each value in x
for i = 1:n
 w = wfun(h,x(i),x);
 yhatnw(i) = sum(w.*y)/sum(w);
end
```

The smooth from the Nadarya-Watson estimator is shown in Figure 10.10. ❑

#### Local Linear Kernel Estimator

When we fit a straight line at a point $x$, then we are using a local linear estimator. This corresponds to the case where $d = 1$, so our estimate is obtained as the solutions $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimize the following,

$$\sum_{i=1}^{n} K_h(X_i - x)(Y_i - \beta_0 - \beta_1(X_i - x))^2.$$

We give an explicit formula for the estimator below.

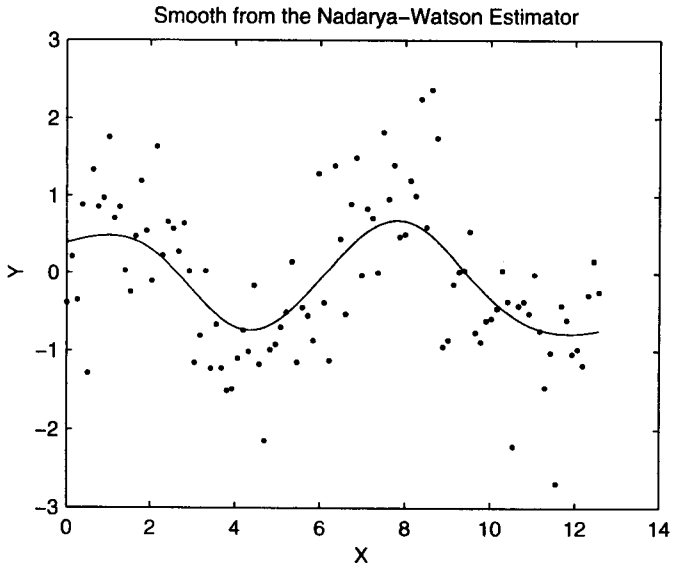



FIGURE 10.10
This figure shows the smooth obtained from the Nadarya-Watson estimator with $h = 1$.

*LOCAL LINEAR KERNEL ESTIMATOR:*

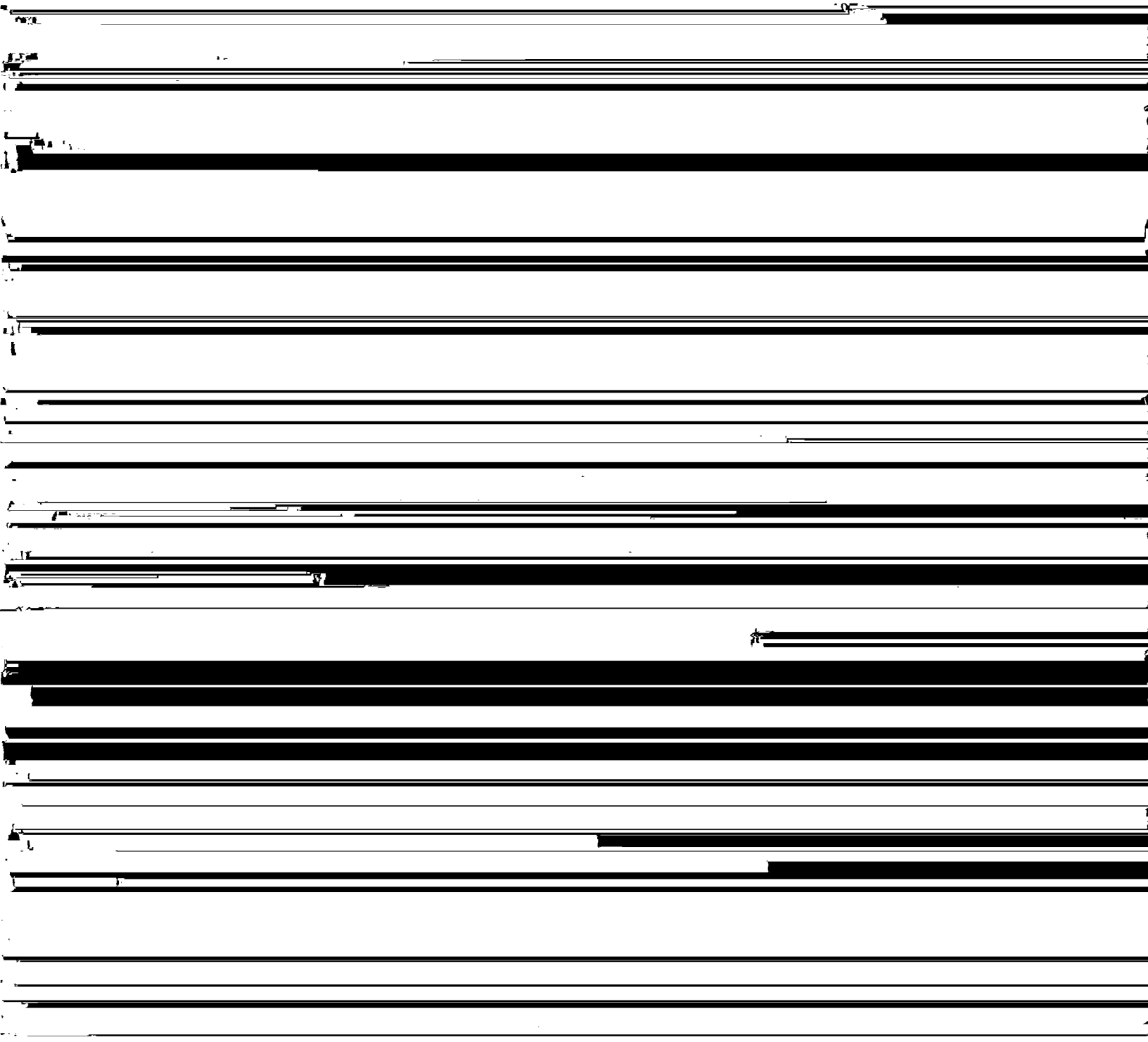
$$\hat{f}_{LL}(x) = \frac{1}{n}\sum_{i=1}^{n} \frac{\{\hat{s}_2(x) - \hat{s}_1(x)(X_i - x)\}K_h(X_i - x)Y_i}{\hat{s}_2(x)\hat{s}_0(x) - \hat{s}_1(x)^2}, \tag{10.24}$$

where

$$\hat{s}_r(x) = \frac{1}{n}\sum_{i=1}^{n} (X_i - x)^r K_h(X_i - x).$$

As before, the fixed design case is obtained by replacing the random variable $X_i$ with the fixed point $x_i$.

When using the kernel smoothing methods, problems can arise near the boundary or extreme edges of the sample. This happens because the kernel window at the boundaries has missing data. In other words, we have weights from the kernel, but no data to associate with them. Wand and Jones [1995] show that the local linear estimator behaves well in most cases, even at the



partitioned at $x_2 = 0.48$.

high, yielding a higher rate at which we accept candidate points. The problem here is that the chain will mix slowly, meaning that the chain will take longer to get to the stationary distribution. On the other hand, if the proposal distribution generates large steps, then the chain could move to the tails, resulting in low acceptance probabilities. Again, the chain fails to mix quickly.

### Example 11.3

In this example, we show how to implement the random-walk version of the Metropolis-Hastings sampler [Gilks, et al., 1996a] and use it to generate variates from the standard normal distribution (the target distribution). Of course, we do not have to resort to MCMC methods to generate random variables from this target distribution, but it serves to illustrate the importance of picking the right scale for the proposal distribution. We use the normal as a proposal distribution to generate the candidates for the next value in the chain. The mean of the proposal distribution is given by the current value in the chain $x_t$. We generate three chains with different values for the standard deviation, given by: $\sigma = 0.5, 0.1, 10$. These provide chains that exhibit good mixing, poor mixing due to small step size and poor mixing due to a large step size, respectively. We show below how to generate the three sequences with $n = 500$ variates in each chain.

```
% Get the variances for the proposal distributions.
sig1 = 0.5;
sig2 = 0.1;
sig3 = 10;
% We will generate 500 iterations of the chain.
n = 500;
% Set up the vectors to store the samples.
X1 = zeros(1,n);
X2 = X1;
X3 = X1;
% Get the starting values for the chains.
X1(1) = -10;
X2(1) = 0;
X3(1) = 0;
```

Now that we have everything initialized, we can obtain the chains.

```
% Run the first chain.
for i = 2:n
 % Generate variate from proposal distribution.
 y = randn(1)*sig1 + X1(i-1);
 % Generate variate from uniform.
 u = rand(1);
 % Calculate alpha.
```

```
 alpha = normpdf(y,0,1)/normpdf(X1(i-1),0,1);
 if u <= alpha
   % Then set the chain to the y.
   X1(i) = y;
 else
   X1(i) = X1(i-1);
 end
end
% Run second chain.
for i = 2:n
 % Generate variate from proposal distribution.
 y = randn(1)*sig2 + X2(i-1);
 % Generate variate from uniform.
 u = rand(1);
 % Calculate alpha.
 alpha = normpdf(y,0,1)/normpdf(X2(i-1),0,1);
 if u <= alpha
   % Then set the chain to the y.
   X2(i) = y;
 else
   X2(i) = X2(i-1);
 end
end
% Run the third chain.
for i = 2:n
 % Generate variate from proposal distribution.
 y = randn(1)*sig3 + X3(i-1);
 % Generate variate from uniform.
 u = rand(1);
 % Calculate alpha.
 alpha = normpdf(y,0,1)/normpdf(X3(i-1),0,1);
 if u <= alpha
   % Then set the chain to the y.
   X3(i) = y;
 else
   X3(i) = X3(i-1);
 end
end
```

Plots of these sequences are illustrated in Figure 11.3, where we also show horizontal lines at $\pm 2$. These lines are provided as a way to determine if most values in the chain are mixing well (taking on many different values) within two standard deviations of zero, since we are generating standard normal variates. Note that the first one converges quite rapidly and exhibits good mixing in spite of an extreme starting point. The second one with $\sigma = 0.1$ (small steps) is mixing very slowly and does not seem to have converged to

the target distribution in these 500 steps of the chain. The third sequence, where large steps are taken, also seems to be mixing slowly, and it is easy to see that the chain sometimes does not move. This is due to the large steps taken by the proposal distribution.
❑

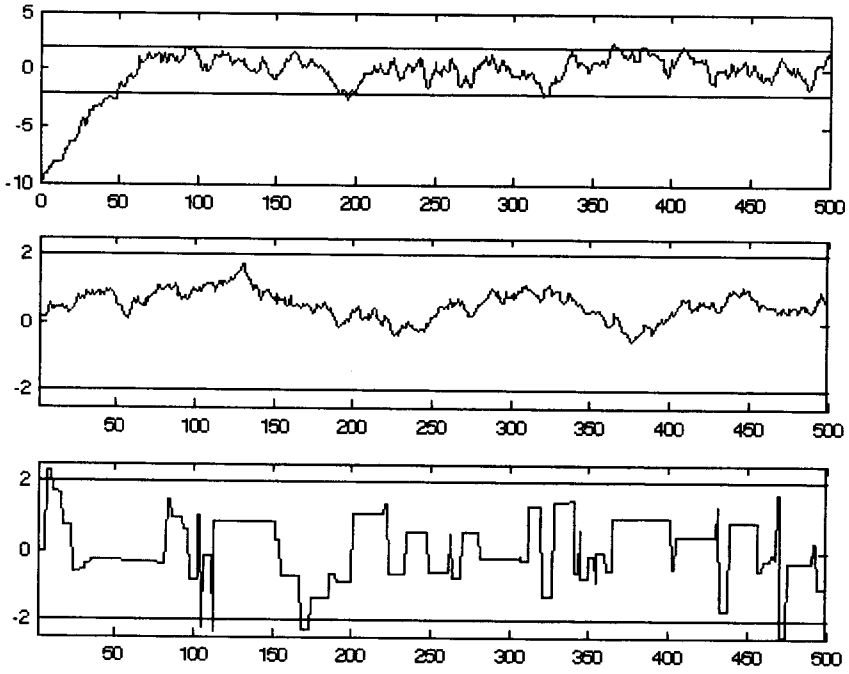



FIGURE 11.3
These are the three sequences from Example 11.3. The target distribution is the standard normal. For all three sequences, the proposal distribution is normal with the mean given by the previous element in the sequence. The standard deviations of the proposal distribution are: $\sigma = 0.5, 0.1, 10$. Note that the first sequence approaches the target distribution after the first 50 - 100 iterations. The other two sequences are slow to converge to the target distribution because of slow mixing due to the poor choice of $\sigma$.

### Independence Sampler

The independence sampler was proposed by Tierney [1994]. This method uses a proposal distribution that does not depend on $\boldsymbol{X}$; i.e., it is generated independently of the previous value in the chain. The proposal distribution is of the form $q(Y|\boldsymbol{X}) = q(Y)$, so Equation 11.6 becomes

$$\alpha(\boldsymbol{X}_t, Y) = \min\left\{1, \frac{\pi(Y)q(\boldsymbol{X}_t)}{\pi(\boldsymbol{X}_t)q(Y)}\right\}. \tag{11.8}$$

This is sometimes written in the literature as

$$\alpha(\boldsymbol{X}_t, Y) = \min\left\{1, \frac{w(Y)}{w(\boldsymbol{X}_t)}\right\}$$

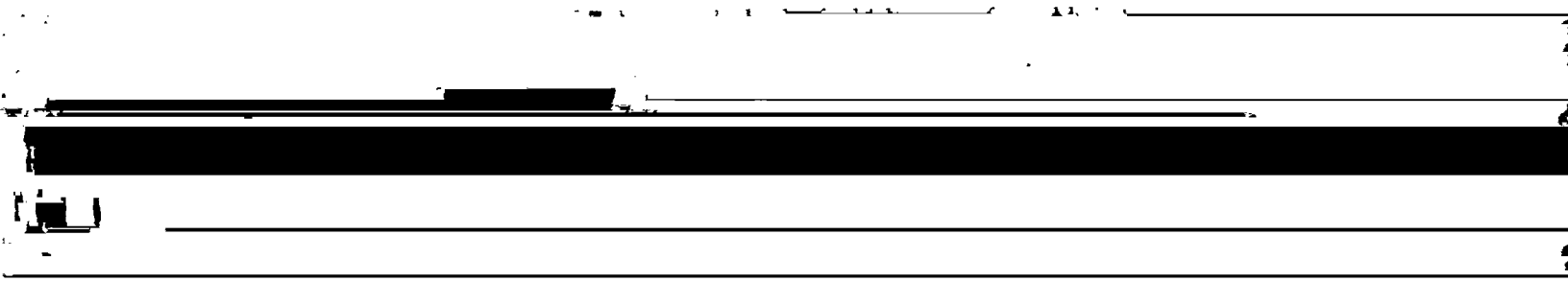
where $w(\boldsymbol{X}) = \pi(\boldsymbol{X})/q(\boldsymbol{X})$.

Caution should be used when implementing the independence sampler. In general, this method will not work well unless the proposal distribution $q$ is very similar to the target distribution $\pi$. Gilks, et al. [1996a] show that it is best if $q$ is heavier-tailed than $\pi$. Note also that the resulting sample is still not independent, even though we generate the candidate points independently of the previous value in the chain. This is because the acceptance probability for the next value $\boldsymbol{X}_{t+1}$ depends on the previous one. For more information on the independence sampler and the recommended usage, see Roberts [1996] or Robert and Casella [1999].

### Autoregressive Generating Density

Another choice for a candidate generating density is proposed by Tierney [1994] and described by Chib and Greenberg [1995]. This is represented by an autoregressive process of order 1 and is obtained by generating candidates as follows

$$\boldsymbol{Y} = \boldsymbol{a} + \mathbf{B}(\boldsymbol{X}_t - \boldsymbol{a}) + \mathbf{Z}, \tag{11.9}$$



Casella and George [1992] and Gelfand and Smith [1990] recommend that $K$ different sequences be generated, each one with length $n$. Then the last element of each sequence is used to obtain a sample of size $K$ that is approximately independent for large enough $K$. We do note that there is some disagreement in the literature regarding the utility of running one really long chain to get better convergence to the target distribution or many shorter chains to get independent samples [Gilks, et al., 1996b]. Most researchers in this field observe that one long run would often be used for exploratory analysis and a few moderate size runs is preferred for inferences.

The procedure given below for the general Gibbs sampler is for one chain only. It is easier to understand the basic concepts by looking at one chain, and it is simple to expand the algorithm to multiple chains.

the model to predict values of the variable at locations where measurements were not taken. Some examples of geostatistical data analysis include the following:

- Rainfall is recorded at various points in a region. These data could be used to model the rainfall over the entire region.
- Geologists take ore samples at locations in a region. They would like to use these data to estimate the extent of the mineral deposit over the entire region.
- Environmentalists measure the level of a pollutant at locations in a region with the goal of using these data to model and estimate the level of pollutant at other locations in the region.

The third type of spatial data is called ***lattice data***. These data are often associated with areas that can be regularly or irregularly spaced. The objective of the analysis of lattice data is to model the spatial pattern in the attributes associated with the fixed areas. Some examples of lattice data are:

- A sociologist has data that comprises socio-economic measures for regions in China. The goal of the analysis might be to describe and to understand any patterns of inequality between the areas.
- Market analysts use socio-economic data from the census to target a promising new area to market their products.
- A political party uses data representing the geographical voting patterns in a previous election to determine a campaign schedule for their candidate.

### Spatial Point Patterns

In this text, we look at techniques for analyzing spatial point patterns only. A spatial point pattern is a set of point locations $\mathbf{s}_1, \ldots, \mathbf{s}_n$ in a study region $\boldsymbol{R}$. Each point location $\mathbf{s}_i$ is a vector containing the coordinates of the $i$-th event,

$$\mathbf{s}_i = \begin{bmatrix} s_{i1} \\ s_{i2} \end{bmatrix}.$$

The term ***event*** can refer to any spatial phenomenon that occurs at a point location. For example, events can be locations of trees growing in a forest, positions of cells in tissue or the incidence of disease at locations in a community. Note that the scale of our study affects the reasonableness of the assumption that the events occur at point locations.

In our analysis of spatial point patterns, we might have to refer to other locations in the study region $\boldsymbol{R}$, where the phenomenon was not observed.

We need a way to distinguish them from the locations where observations were taken, so we refer to these other locations as ***points*** in the region.

At the simplest level, the data we are analyzing consist only of the coordinate locations of the events. As mentioned before, they could also have an attribute or variable associated with them. For example, this attribute might be the date of onset of the disease, the species of tree that is growing, or the type of crime.

This type of spatial data is sometimes referred to as a ***marked point pattern***. In our treatment of spatial point patterns, we assume that the data represent a ***mapped point pattern***. This is one where all relevant events in the study region $\boldsymbol{R}$ have been measured. The study region $\boldsymbol{R}$ can be any shape. However, edge effects can be a problem with many methods in spatial statistics. We describe the ramifications of edge effects as they arise with the various techniques. In some cases, edge effects are handled by leaving a specified guard area around the edge of the study region, but still within $\boldsymbol{R}$. The analysis of point patterns is sensitive to the definition of $\boldsymbol{R}$, so one might want to perform the analysis for different guard areas and/or different study regions.

One way we can think of spatial point patterns is in terms of the number of events occurring in an arbitrary sub-region of $\boldsymbol{R}$. We denote the number of events in a sub-region $\boldsymbol{A}$ as $Y(\boldsymbol{A})$. The spatial process is then represented by the random variables $Y(\boldsymbol{A})$, $\boldsymbol{A} \subset \boldsymbol{R}$. Since we have a random process, we can look at the behavior in terms of the first-order and second-order properties. These are related to the expected value (i.e., the mean) and the covariance [Bailey and Gatrell, 1995]. The mean and the covariance of $Y(\boldsymbol{A})$ depend on the number of events in arbitrary sub-regions $\boldsymbol{A}$, and they depend on the size of the areas and the study region $\boldsymbol{R}$. Thus, it is more useful to look at the first- and second-order properties in terms of the limiting behavior per unit area.

The first-order property is described by the intensity $\lambda(\mathbf{s})$. The ***intensity*** is defined as the mean number of events per unit area at the point $\mathbf{s}$. Mathematically, the intensity is given by

$$\lambda(\mathbf{s}) = \lim_{ds \to 0} \left\{ \frac{E[Y(\mathbf{ds})]}{ds} \right\}, \tag{12.1}$$

where $\mathbf{ds}$ is a small region around the point $\mathbf{s}$, and $ds$ is its area. If it is a ***stationary point process***, then Equation 12.1 is a constant over the study region. We can then write the intensity as

$$E[Y(\boldsymbol{A})] = \lambda A, \tag{12.2}$$

where $A$ is the area of the sub-region, and $\lambda$ is the value of the intensity.

To understand the second-order properties of a spatial point process, we need to look at the number of events in pairs of sub-regions of $\boldsymbol{R}$. The second-order property reflects the spatial dependence in the process. We describe

this using the second-order intensity $\gamma(\mathbf{s}_i, \mathbf{s}_j)$. As with the intensity, this is defined using the events per unit area, as follows,

$$\gamma(\mathbf{s}_i, \mathbf{s}_j) = \lim_{ds_i, ds_j \to 0} \left\{ \frac{E[Y(\mathbf{ds}_i)Y(\mathbf{ds}_j)]}{ds_i, ds_j} \right\}. \tag{12.3}$$

If the process is stationary, then $\gamma(\mathbf{s}_i, \mathbf{s}_j) = \gamma(\mathbf{s}_i - \mathbf{s}_j)$. This means that the second-order intensity depends only on the vector difference of the two points. The process is said to be ***second-order*** and ***isotropic*** if the second-order intensity depends only on the distance between $\mathbf{s}_i$ and $\mathbf{s}_j$. In other words, it does not depend on the direction.

### Complete Spatial Randomness

The benchmark model for spatial point patterns is called ***complete spatial randomness*** or **CSR**. In this model, events follow a homogeneous Poisson process over the study region. The definition of CSR is given by the following [Diggle, 1983]:

1. The intensity does not vary over the region. Thus, $Y(A)$ follows a Poisson distribution with mean $\lambda A$, where $A$ is the area of $A$ and $\lambda$ is constant.
2. There are no interactions between the events. This means that, for a given $n$, representing the total number of events in $R$, the events are uniformly and independently distributed over the study region.

In a CSR process, an event has the same probability of occurring at any location in $R$, and events neither inhibit nor attract each other. The methods covered in this chapter are mostly concerned with discovering and modeling

```
lam = 15;
% Get the number of children per parent.
nchild = poissrnd(lam,1,npar);
```

Now we find the locations of the children around the parent using a bivariate normal distribution that is centered at each parent. The covariance of the distribution is given by $\sigma^2\mathbf{I}$, where $\mathbf{I}$ is a $2 \times 2$ identity matrix. The value given to the variance $\sigma^2$ would govern the spread of the cluster of children around the parent.

```
X = [];
sig = r*eye(2);
r = 0.05;
% Locate the children.
for i = 1:npar
   xc = randn(nchild(i),2)*sig + ...
     repmat([xp(i) yp(i)],nchild(i),1);
   X = [X; xc];
end
```

To get the final events for our sample, we need to determine which ones are inside the study region $R$. We do this using the MATLAB function **inpolygon**. In Figure 12.19, we show the resulting spatial sample. We provide a function called **csclustproc** that will generate patterns that follow a Poisson cluster process.

```
% Find the ones that are in the region of interest.
ind = find(inpolygon(X(:,1), X(:,2), rx, ry));
% Those are the children for the sample.
x = X(ind,1);
y = X(ind,2);
```

❑

### Inhibition Process

An inhibition process is one that often shows regularity. To simulate this type of process, we include a mechanism in the model that stipulates a minimum distance between two events. We call this distance the ***inhibition distance*** $\delta$.

One way to obtain such a process is to first generate a homogeneous Poisson process over the region. The events are then thinned by deleting all pairs of events that are closer than $\delta$. Implementing this procedure in MATLAB is left as an exercise.

Another method is to generate a homogeneous Poisson process one event at a time and discard candidate events if they are within distance $\delta$ of any previously retained event. This type of process is sometimes referred to as ***Sequential Spatial Inhibition*** or SSI [Ripley, 1981]. It is important to keep in mind that if the inhibition distance is too large for the region $R$, then it might

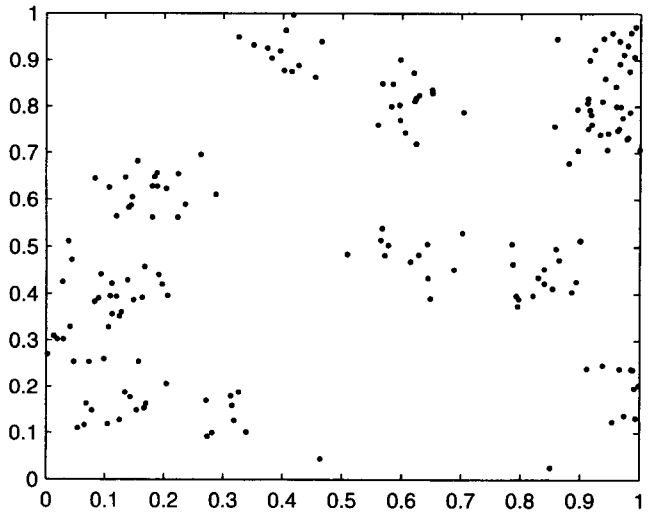



FIGURE 12.19
This sample was generated according to a Poisson cluster process.

be difficult (if not impossible) to generate the required number of points. In Example 12.13, we provide the MATLAB code to generate an inhibition spatial point pattern using this procedure.

## Example 12.13

To start the procedure, we set the boundary for the region and the inhibition distance.

```
delta = 0.1;
% Get the vertices for the regions.
rx = [0 2 2 0 0];
ry = [0 0 2 2 0];
n = 100;
```

We generate the initial event from a CSR process. Subsequent events are generated and kept if they are not closer than $\delta$ to any existing events.

```
X = zeros(n,2);
% Generate the first event.
X(1,:) = csbinproc(rx,ry,1);
i = 1;
% Generate the other events.
while i<n
```

```
[sx,sy] = csbinproc(rx, ry, 1);
xt = [sx sy ; X(1:i,:)];
% Find the distance between the events
dist = pdist(xt);
% Find the distance between the candidate event
% and the others that have been generated already.
ind = find(dist(1:i) <= delta);
if isempty(ind)
  % Then we keep the event.
  i = i+1;
  X(i,:) = [sx, sy];
end
end
```

To verify that no two events are closer than $\delta$, we find the smallest distance as follows.

```
% Verify that all are no closer than the
% inhibition distance.
dist = pdist(X);
delhat = min(dist);
```

For this spatial point pattern, we get a minimum distance of 0.1008. A point pattern generated according to this procedure is shown in Figure 12.20. ❑

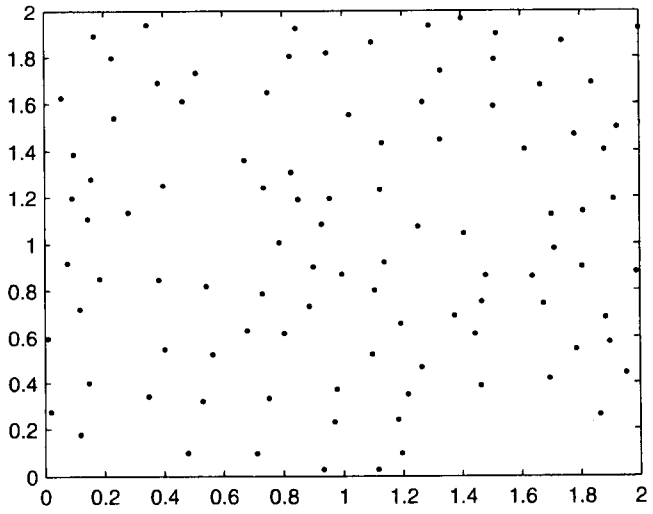



FIGURE 12.20
This spatial point pattern was generated under the SSI inhibition process.

### Strauss Process

The Strauss process [Ripley, 1981] is a point pattern where a specified fraction of events is allowed within a distance $\delta$ of any given event. To generate such a pattern, the first event is located uniformly in $\boldsymbol{R}$. Other event locations are generated sequentially, similar to the SSI process. If there are existing events within radius $\delta$ of the candidate location, then it is accepted with probability $c^m$, with $m$ representing the number of events closer than $\delta$. The ***inhibition parameter*** is given by $c$, which can take on values in the interval $[0, 1]$.

The inhibition parameter specifies the fraction of events allowed within the inhibition distance. If $c = 0$, then the resulting process is the same as SSI. As with the SSI process, care should be taken when specifying the parameters for the process to ensure that the required number of events can be generated. We outline below the steps to generate a spatial point pattern that follows a Strauss process.

*PROCEDURE - SIMULATING A STRAUSS PROCESS*

1. Choose the parameters $n$, $c$, and $\delta$.
2. Generate the first event location $\boldsymbol{s}_1$ uniformly on $\boldsymbol{R}$ (from a CSR process).
3. Generate a candidate location $\boldsymbol{s}_i$ uniformly on $\boldsymbol{R}$.
4. If $m = 0$

   accept the candidate event $\boldsymbol{s}_i$

   Else if $U \leq c^m$

   accept the candidate event $\boldsymbol{s}_i$
5. Repeat steps 3 and 4 until there are $n$ locations in the sample.

It should be noted that we are conditioning on the number of points $n$ in the region. So, in this case, we should consider this a conditional Strauss process.

### Example 12.14

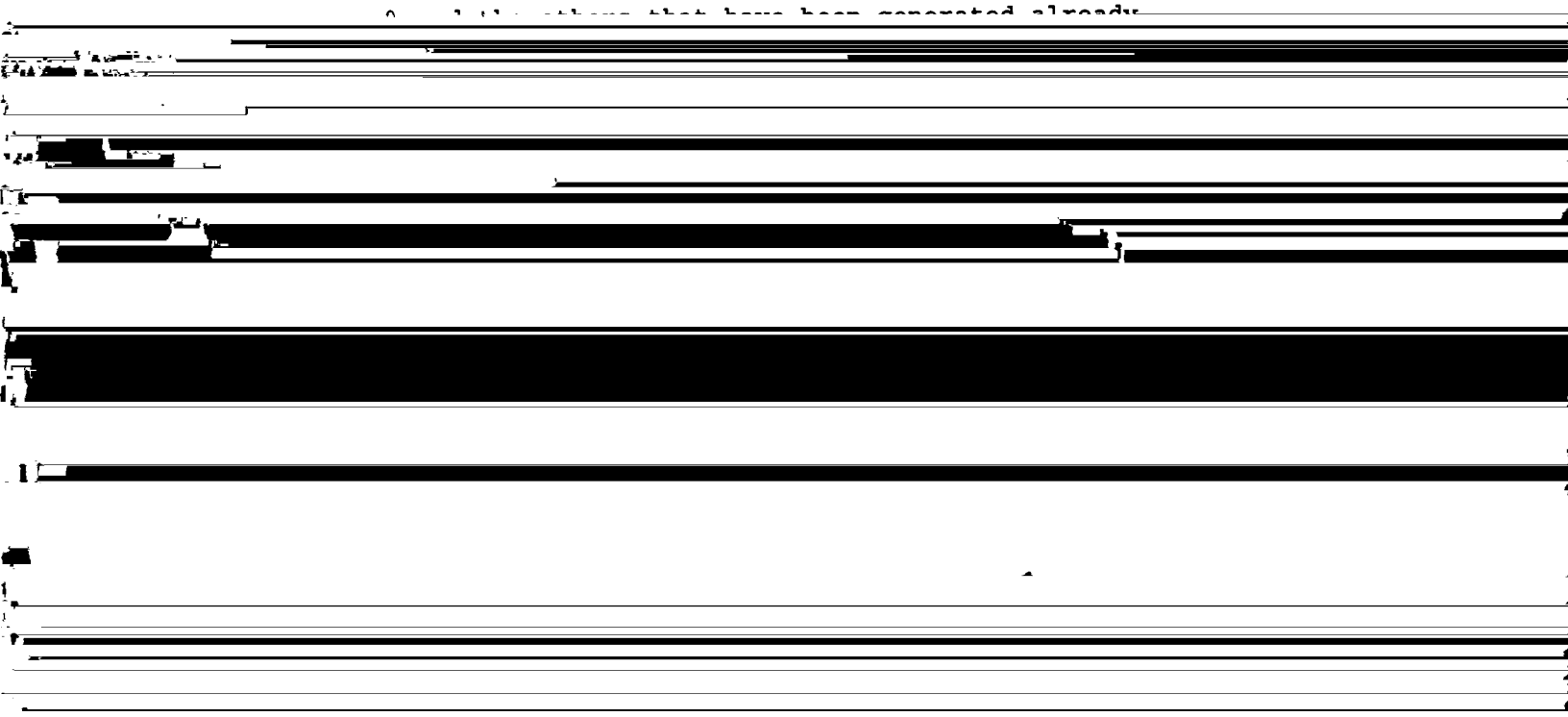
We now implement the above procedure in MATLAB. We generate a spatial point pattern of size 100 from a Strauss process over a rectangular region. The inhibition distance is $\delta = 0.1$, and the inhibition parameter is $c = 0.5$. We start by setting these parameters and the boundary of the study region.

```
delta = 0.1;
% Get the vertices for the regions.
rx = [0 1 1 0 0];
ry = [0 0 2 2 0];
% Set number of data points.
n = 100;
% Set the inhibition parameter.
```

```
c = 0.5;
X = zeros(n,2);
% Generate the first point.
X(1,:) = csbinproc(rx,ry,1);
```

The following code is similar to the SSI process, except that we now have a mechanism for accepting points that are closer than the inhibition distance.

```
i = 1;
while i<n
 [sx,sy] = csbinproc(rx, ry, 1);
 xt = [sx sy ; X(1:i,:)];
 % Find the distance between the events.
 dist = pdist(xt);
 % Find the distance between the candidate event
```



up-arrow and down-arrow keys scroll through the commands. The left and right arrows move through the present command. By using these keys, the user can recall commands and edit them using common editing keystrokes.

We can view the contents of the current workspace using the **Workspace Browser**. This is accessed through the **File** menu or the toolbar. All variables in the workspace are listed in the window. The variables can be viewed

```
tree.node.impurity = sqrer(y,tree.n);
tree.node.parent = 0;
tree.node.children = [];
tree.node.sibling = [];
tree.node.yhat = mean(y);
tree.node.split = [];
tree.node.var = [];
tree.node.x = X;
tree.node.y = y;

% Now get started on growing the tree very large
[term,nt,imp]=getdata(tree);
% find all of the nodes that qualify for splitting
ind = find( (term==1) & (imp>0) & (nt>maxn) );
% now start splitting
while ~isempty(ind)
   for i=1:length(ind)
      % get split
      [split,dim]=splitnoder(...
         tree.node(ind(i)).x,...
         tree.node(ind(i)).y,...
         tree.node(ind(i)).impurity,...
         tree.n);
      % split the node
      tree = addnoder(tree,ind(i),dim,split);
   end  % end for loop
   [term,nt,imp]=getdata(tree);
   tree.termnodes = find(term==1);
   ind = find( (term==1) & (imp>0) & (nt>maxn) );
end  % end while loop
```

# Appendix E

## *MATLAB Statistics Toolbox*

The following tables list the functions that are available in the MATLAB Statistics Toolbox, Version 3.0. This toolbox is available for purchase from The MathWorks, Inc.

**TABLE E.1**

Functions for Parameter Estimation (`fit`) and Distribution Statistics - Mean and Variance (`stat`)

| Function | Purpose |
|---|---|
| `betafit, betastat` | Beta distribution. |
| `binofit, binostat` | Binomial distribution. |
| `expfit, expstat` | Exponential distribution. |
| `fstat` | F distribution |
| `gamfit, gamstat` | Gamma distribution. |
| `geostat` | Geometric distribution |
| `hygestat` | Hypergeometric distribution |
| `lognstat` | Lognormal distribution |
| `mle` | Maximum likelihood parameter estimation. |
| `nbinstat` | Negative binomial distribution |
| `ncfstat` | Noncentral F distribution |
| `nctstat` | Noncentral t distribution |
| `ncx2stat` | Noncentral Chi-square distribution |
| `normfit, normstat` | Normal distribution. |
| `poissfit, poisstat` | Poisson distribution. |
| `raylfit` | Rayleigh distribution. |
| `tstat` | T distribution |
| `unidstat` | Discrete uniform distribution |
| `unifit, unifstat` | Uniform distribution. |
| `weibfit, weibstat` | Weibull distribution. |

TABLE E.2

Probability Density Functions (**pdf**) and Cumulative Distribution Functions (**cdf**)

| Function | Purpose |
|---|---|
| `betapdf, betacdf` | Beta distribution |
| `binopdf, binocdf` | Binomial distribution |
| `chi2pdf, chi2cdf` | Chi-square distribution |
| `exppdf, expcdf` | Exponential distribution |
| `fpdf, fcdf` | F distribution |
| `gampdf, gamcdf` | Gamma distribution |
| `geopdf, geocdf` | Geometric distribution |
| `hygepdf, hygecdf` | Hypergeometric distribution |
| `lognpdf, logncdf` | Log normal distribution |
| `nbinpdf, nbincdf` | Negative binomial distribution |
| `ncfpdf, ncfcdf` | Noncentral F distribution |
| `nctpdf, nctcdf` | Noncentral t distribution |
| `ncx2pdf, ncx2cdf` | Noncentral chi-square distribution |
| `normpdf, normcdf` | Normal distribution |
| `pdf, cdf` | Probability density/Cumulative distribution |
| `poisspdf, poisscdf` | Poisson distribution |
| `raylpdf, raylcdf` | Rayleigh distribution |
| `tpdf, tcdf` | T distribution |
| `unidpdf, unidcdf` | Discrete uniform distribution |
| `unifpdf, unifcdf` | Continuous uniform distribution |
| `weibpdf, weibcdf` | Weibull distribution |

TABLE E.3

Critical Values (**inv**) and Random Number Generation (**rnd**) for Probability Distribution Functions

| Function | Purpose |
|---|---|
| `betainv, betarnd` | Beta distribution |
| `binoinv, binornd` | Binomial distribution |
| `chi2inv, chi2rnd` | Chi-square distribution |
| `expinv, exprnd` | Exponential distribution |
| `finv, frnd` | F distribution |
| `gaminv, gamrnd` | Gamma distribution |
| `geoinv, geornd` | Geometric distribution |
| `hygeinv, hygernd` | Hypergeometric distribution |
| `logninv, lognrnd` | Log normal distribution |
| `nbininv, nbinrnd` | Negative binomial distribution |
| `ncfinv, ncfrnd` | Noncentral F distribution |
| `nctinv, nctrnd` | Noncentral t distribution |
| `ncx2inv, ncx2rnd` | Noncentral chi-square distribution |
| `norminv, normrnd` | Normal distribution |
| `poissinv, poissrnd` | Poisson distribution |
| `raylinv, raylrnd` | Rayleigh distribution |
| `tinv, trnd` | T distribution |
| `unidinv, unidrnd` | Discrete uniform distribution |
| `unifinv, unifrnd` | Continuous uniform distribution |
| `weibinv, weibrnd` | Weibull distribution |
| `icdf` | Specified inverse cdf |

TABLE E.4

Descriptive Statistics

| Function | Purpose |
|---|---|
| `bootstrp` | Bootstrap statistics for any function. |
| `corrcoef` | Correlation coefficient - also in standard MATLAB |
| `cov` | Covariance - also in standard MATLAB |
| `crosstab` | Cross tabulation |
| `geomean` | Geometric mean |
| `grpstats` | Summary statistics by group |
| `harmmean` | Harmonic mean |
| `iqr` | Interquartile range |
| `kurtosis` | Kurtosis |
| `mad` | Median absolute deviation |
| `mean` | Sample average - also in standard MATLAB |
| `median` | Second quartile (50th percentile) of a sample - also in standard MATLAB |
| `moment` | Moments of a sample |
| `nanmax, nanmin` | Maximum/minimum - ignoring NaNs |
| `nanmean, nanmedian` | Mean/median - ignoring NaNs |
| `nanstd, namsum` | Standard deviation/sum - ignoring NaNs |
| `prctile` | Percentiles |
| `range` | Range |
| `skewness` | Skewness |
| `std` | Standard deviation - also in standard MATLAB |
| `tabulate` | Frequency table |
| `trimmean` | Trimmed mean |
| `var` | Variance - also in standard MATLAB |

TABLE E.5

Linear Models

| Function | Purpose |
|---|---|
| `anova1` | One-way analysis of variance |
| `anova2` | Two-way analysis of variance |
| `anovan` | n-way analysis of variance |
| `aoctool` | Interactive tool for analysis of covariance |
| `dummyvar` | Dummy-variable coding |
| `friedman` | Friedman's test |
| `glmfit` | Generalized linear model fitting |
| `kruskalwallis` | Kruskal-Wallis test |
| `lscov` | Least-squares estimates with known covariance matrix |
| `manoval` | One-way multivariate analysis of variance |
| `manovacluster` | Draw clusters of group means for manova1 |
| `multcompare` | Multiple comparisons of means and other estimates |
| `polyconf` | Polynomial evaluation and confidence interval estimation |
| `polyfit` | Least-squares polynomial fitting- also in standard MATLAB |
| `polyval` | Predicted values for polynomial functions- also in standard MATLAB |
| `rcoplot` | Residuals case order plot |
| `regress` | Multivariate linear regression |
| `regstats` | Regression diagnostics |
| `ridge` | Ridge regression |
| `robustfit` | Robust regression model fitting |
| `rstool` | Multidimensional response surface visualization |
| `stepwise` | Interactive tool for stepwise regression |
| `x2fx` | Factor setting matrix (x) to design matrix (fx) |

TABLE E.6
Nonlinear Models

| Function | Purpose |
|---|---|
| `nlinfit` | Nonlinear least-squares data fitting (Newton's Method) |
| `nlintool` | Interactive graphical tool for prediction in nonlinear models |

bility to radial highways, (10) full-value property-tax rate per $10,000, (11)

Spath, Helmuth. 1980. *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*, New York: Halsted Press.

Strang, Gilbert. 1988. *Linear Algebra and its Applications*, Third Edition, San Diego: Harcourt Brace Jovanovich.

Swayne, D. F., D. Cook, and A. Buja. 1991. "XGobi: Interactive dynamic graphics in the X window system with a link to S," *ASA Proceedings of the Section on Statistical Graphics*. pp. 1-8.

Tanner, Martin A. *Tools for Statistical Inference: Methods for the Exploration of Posterior Distributions and Likelihood Functions, Third Edition*, New York: Springer-Verlag.

Tapia, R. A. and J. R. Thompson. 1978. *Nonparametric Probability Density Estimation*, Baltimore: Johns Hopkins University Press.

Teichroew, D. 1965. "A history of distribution sampling prior to the era of the computer and its relevance to simulation," *Journal of the American Statistical Association*, **60**: pp. 27-49.

Terrell, G. R. 1990. "The maximal smoothing principle in density estimation," *Journal of the American Statistical Association*, **85**: p. 470-477.

Thisted, R. A. 1988. *Elements of Statistical Computing*, London: Chapman and Hall.

Tibshirani, R. 1988. "Variance stabilization and the bootstrap," *Biometrika*, **75**: pp. 433-444.

Tierney, L. 1994. "Markov chains for exploring posterior distributions (with discussion)," *Annals of Statistics*, **22**: pp. 1701-1762.

Tierney, L. 1996. "Introduction to general state-space Markov chain theory," in *Markov Chain Monte Carlo in Practice*, W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, eds., London: Chapman and Hall, pp. 59-74.

Tinkler, K. J. 1971. "Statistical analysis of tectonic patterns in areal volcanism: the Bunyaruguru volcanic field in west Uganda," *Mathematical Geology*, **3**: pp. 335–355.

Titterington, D. M., A. F. M. Smith, and U. E. Makov. 1985. *Statistical Analysis of Finite Mixture Distributions*, New York: John Wiley & Sons.

Tripathi, R. C. and R. C. Gupta. 1988. "Another generalization of the logarithmic series and the geometric distribution," *Communications in Statistics - Theory and Methods*, **17**: pp. 1541-1547.

Tufte, E. 1983. *The Visual Display of Quantitative Information*, Cheshire, CT: Graphics Press.

Tufte, E. 1990. *Envisioning Information*, Cheshire, CT: Graphics Press.

Tufte, E. 1997. *Visual Explanations*, Cheshire, CT: Graphics Press.

Tukey, John W. 1958. "Bias and confidence in not quite large samples," *Annals of Mathematical Statistics*, **29**: pp. 614.

Tukey, John W. 1977. *Exploratory Data Analysis*, New York: Addison-Wesley.

Upton, G. and B. Fingleton. 1985. *Spatial Data Analysis by Example: Volume I: Point Pattern and Quantitative Data*, New York: John Wiley & sons.

Utts, Jessica. 1996. *Seeing Through Statistics*, New York: Duxbury Press.

van Oost, B. A., B. Veldhayzen, A. P. M. Timmermans, and J. J. Sixma. 1983. "Increased urinary $\beta$-thromoglobulin excretion in diabetes assayed with a modified RIA kit-technique," *Thrombosis and Haemostasis*, **9**: pp. 18-20.

Venables, W. N. and B. D. Ripley. 1994. *Modern Applied Statistics with S-Plus*, Ne York: Springer-Verlag.

Wadsworth, H. M. (ed.). 1990. *Handbook of Statistical Methods for Engineers and Scie tists*, New York: McGraw-Hill.

Wainer, H. 1997. *Visual Revelations: Graphical Tales of Fate and Deception from Napoleo Bonaparte to Ross Perot*, New York: Copernicus/Springer-Verlag.

Walpole, R. E. and R. H. Myers. 1985. *Probability and Statistics for Engineers an Scientists*, New York: Macmillan Publishing Company.

Wand, M.P. and M. C. Jones. 1995. *Kernel Smoothing*, London: Chapman and Hall.

Watson, G. S. 1964. "Smooth regression analysis," *Sankhya Series A*, **26**: pp. 101-116.

Webb, Andrew. 1999. *Statistical Pattern Recognition*, Oxford: Oxford University Press

Wegman, E. 1986. *Hyperdimensional Data Analysis Using Parallel Coordinates*, Technical Report No. 1, George Mason University Center for Computational Statistics.

Wegman, E. 1988. "Computational statistics: A new agenda for statistical theory and practice," *Journal of the Washington Academy of Sciences*, **78**: pp. 310-322.

Wegman, E. 1990. "Hyperdimensional data analysis using parallel coordinates," *Journal of the American Statistical Association*, **85**: pp. 664-675.

Wegman, E. and J. Shen. 1993. "Three-dimensional Andrews plots and the grand tour," *Proceedings of the 25th Symposium on the Interface*, pp. 284-288.

Wegman, E., D. Carr, and Q. Luo. 1993. "Visualizing multivariate data," in *Multivariate Analysis: Future Directions*, C. R. Rao, ed., The Netherlands: Elsevier Science Publishers, pp. 423-466.

Weiss, Neil. 1999. *Introductory Statistics*, New York: Addison Wesley Longman.

Wilcox, Rand R. 1997. *Introduction to Robust Estimation and Hypothesis Testing*, New York: Academic Press.

Wilk, M. and R. Gnanadesikan. 1968. "Probability plotting methods for the analysis of data," *Biometrika*, **55**: pp. 1-17.

Wilkinson, Leland. 1999. *The Grammar of Graphics*, New York: Springer-Verlag.

# *Index*


## A

adaptive mixtures density
  estimation, **301**, **540**
adjacent values, **133**
agglomerative clustering, **369**
alternative hypothesis, **193**
Andrews curves, **157**, **163**, **164**
aperiodicity, **431**
autoregressive generating
  density, **439**
average linkage, **371**
averaged shifted histogram, **274**

## B

backslash operator, **387**
bagplot, **183**
**bar**, **267**
Bayes Decision Rule, **323**
Bayes decision theory, **319**
Bayes' Theorem, **19**, **320**, **426**
Bayesian inference, **426**
Bernoulli random variable, **24**
**beta**, **40**
beta distribution, **40**
  random number generation, **95**
**betainc**, **41**
**betapdf**, **41**
**betarnd**, **96**, **445**
bias, **61**
**binocdf**, **26**
binomial distribution, **24**
  random number generation, **100**
binomial process, **497**
binomialness plot, **129**
**binopdf**, **26**, **446**
**binornd**, **445**
bisquare method, **396**
**bootfun**, **226**
bootstrap, **214**, **221**
  bias, **219**
  bias corrected interval, **247**, **539**
  confidence interval, **220**
  jackknife-after-bootstrap, **251**
  parametric bootstrap, **216**
  percentile interval, **224**, **247**
  standard error, **216**
  t-interval, **221**
bootstrp, **218**, **223**, **225**, **252**
box plot, **132**, **186**
boxplot, **133**
brushing, **185**
BUGS, **460**
burn-in, **429**

## C

Cauchy distribution, **432**
cdfplot, **76**, **183**
Central Limit Theorem, **59**, **168**
centroid linkage, **371**
**char**, **516**
Chernoff faces, **185**
chi-square distribution, **37**
  random number generation, **93**
chi-square index, **169**
city block distance, **368**
class-conditional probability, **320**
  estimating, **321**
  nonparametric estimation, **322**
classification trees, **342**
  choosing the tree, **356**
  cost complexity measure, **352**
  cross-validation, **361**
  growing the tree, **347**
  impurity, **349**
  independent test sample, **357**
  MATLAB code, **542**
  notation, **345**
  pruning process, **352**
  splitting rule, **349**
clustering, **367**
  hierarchical methods, **369**
  linkage methods, **370**
clustering, spatial, **469**
complete linkage, **370**
complete spatial randomness, **469**
computational statistics, **1**
Computational Statistics Toolbox, **7**
conditional probability, **17**
confidence intervals, **201**

**contour**, 138
contour plots, 138
**contour3**, 138
**contourf**, 138
convergence monitoring
  Gelman and Rubin method, 453
  Raftery and Lewis method, 458
**convhull**, 474
cophenetic correlation
  coefficient, 373
**corr**, 242
**corrcoef**, 57, 241
correlation, 56, 241
**cov**, 57
covariance, 56
covariance matrix, 41
critical region, 194
critical value, 194
cross-validation, 233
  *K*-fold, 237
  pattern recognition, 335
**csadpmix**, 304
**csbinproc**, 482
**csbinrnd**, 101
**cschirnd**, 94
**csclustproc**, 501
**csdfplot**, 294
**csdunrnd**, 89, 105
**csevalnorm**, 43, 325
**csgelrub**, 457
**csgetregion**, 508
**csgrowc**, 351
**csgrowr**, 411, 417
**csintenkern**, 476
**csjack**, 242
**cskernmd**, 323
**cskhat**, 484
**csloess**, 396
**csloessenv**, 401
**csloessr**, 400
**csmvrnd**, 98
**csplotreec**, 351
**cspoirnd**, 103
**csprunec**, 355
**cspruner**, 417
**csquantiles**, 121
**cssphrnd**, 100
**csstemleaf**, 117
**cstreer**, 418
cumulative distribution function, 15

## D

data sets, 563
  **abrasion**, 423
  **anaerob**, 423
  **anscombe**, 243
  **bank**, 189, 381
  **biology**, 188
  **bodmin**, 480, 489, 492
  **boston**, 423
  **brownlee**, 423
  **cardiff**, 472, 484, 493
  **cement**, 57
  **cereal**, 156
  **counting**, 188
  **elderly**, 316
  **environ**, 393, 422
  **filip**, 388, 399, 422
  **flea**, 382
  **forearm**, 113, 115, 217, 220, 222, 225, 228, 316
  **geyser**, 266, 271, 309, 315
  **helmets**, 423
  **household**, 381
  **human**, 423
  **insect**, 146, 319
  **insulate**, 256
  **iris**, 148, 160, 164, 182, 286, 314, 318, 319, 321, 322, 333, 335, 364, 375
  **law**, 241, 245, 252, 257
  **lawpop**, 228
  **longley**, 387
  **measure**, 381
  **moths**, 188
  **nfl**, 316
  **okblack**, 473
  **okwhite**, 473
  **peanuts**, 256
  **quakes**, 229
  **remiss**, 228
  **snowfall**, 278, 314, 315
  **steam**, 234, 236
  **thrombos**, 188
  **tibetan**, 117, 381
  **uganda**, 471, 476, 498
  **whisky**, 229
degree of model, 233
dendrogram, 371
*dF* plot, 291
digidot plot, 185

discrete uniform distribution, random number generation, **104**
dissimilarity, **367**
distance, **367**
**disttool**, 50
divisive clustering, **369**
dot map, **471**
**double**, 516

## E

efficiency, **62**
EM algorithm, **296**
  iterative, **299**
  recursive, **301**
empirical distribution function, **69**
entropy index, **530**
equal likelihood model, **14**
**erf**, 32
error complexity measure, **411**
**errorbar**, 183
estimator, **61**
Euclidean distance, **367**
event, spatial, **467**
expected value, **21**
experiment, **12**
exploratory data analysis, **111**, **259**
**expocdf**, 35
exponential distribution, **33**
  random number generation, **89**
**exprnd**, 428
**eye**, 517

## F

features, **318**
finite mixtures, **287**
  *dF* plot, **291**
  EM algorithm, **296**
  mixing coefficient, weight, **289**
  multivariate, **293**
  posterior probabilities, **297**
frequency polygon, **269**
  asymptotic MISE, **269**
  multivariate, **272**
  normal reference rule, **270**
  skewness factor, **270**
Friedman-Tukey index, **529**

## G

gamma distribution, **36**, **66**
  random number generation, **91**
**gammainc**, 37
**gampdf**, 37
**gamrnd**, 452
geostatistical data, **466**
Gibbs sampler, **443**
Gini index, **349**
grand tour, **178**

## H

**help**, 512
**hist**, 113, 263, 266
**histc**, 266, 278
**histfit**, 76
histogram, **113**, **261**
  asymptotic MISE, **264**
  bin width, **261**
  bivariate, **141**
  density histogram, **114**
  Freedman-Diaconis rule, **265**
  frequency histogram, **113**
  mean squared error, pointwise, **263**
  normal reference rule, **265**
  relative frequency histogram, **113**
  Scott's rule, **265**
  skewness factor, **266**
  Sturge's rule, **264**
histogram, multivariate, **267**
  asymptotic MISE, **268**
  normal reference rule, **268**
hypothesis testing, **192**
  critical value approach, **195**
  Monte Carlo approach, **205**
  *p*-value approach, **200**

## I

iid, **52**
independence sampler, **438**
independent events, **18**
inferential statistics, **52**
inhibition distance, **501**
inhibition parameter, **504**
inhibition process, **501**

**inline**, 278, 405, 432, 442, 519
**inpolygon**, 273, 501
integrated squared error, 260
intensity
  estimation of, 475
  second-order, 469
  spatial, 468
internet sources, 7
**interp1**, 72, 271
**interp2**, 274
interquartile range, 71, 132
irreducibility, 431
**isocaps**, 153
**isosurface**, 149
isotropic, 469

## J

jackknife, 220, 239, 248
  bias, 240
  jackknife-after-bootstrap, 251
  pseudo-values, 243
  standard error, 240
joint probability, 18

## K

*k* nearest neighbors, 383
kernel density estimation, 280
  asymptotic MISE, univariate, 282
  choice of kernel, 284
  multivariate, 285
  normal reference rule, multivariate, 286
  normal reference rule, univariate, 283
  window width, 282
*K*-function, 482, 490
*k*-means clustering, 373
kurtosis, 23, 54

## L

lattice data, 467
least squares, 386
*L*-function, 484, 491
likelihood, 426
likelihood function, 64
limits, lower and upper, 133
linkage, 370
linkage methods, 369
**load**, 514
local linear estimator, 405
local polynomial kernel estimators, 402
loess, 391
  envelopes, 400
  robust, 396
lower tail test, 195

## M

Mahalanobis distance, 302, 367
mapped point pattern, 468
marked point pattern, 468
Markov chain, 429
Markov chain Monte Carlo
  burn-in, 429
  convergence monitoring, 452
MATLAB
  cell arrays, 517
  control flow, 520
  data constructs, 516
  file I/O, 514
  graphing techniques, 521
  **help**, 512
  **inline** function object, 519
  internet resources, 7
  operators, 514
  product information, 524
  programs, 518
  structures, 518
  workspace, 512
maximum likelihood estimation, 63
mean integrated absolute error, 260
mean integrated squared error, 260
mean squared error, 61, 260
median, 70, 246
mesh, 138
method of moments, 66
Metropolis sampler, 433
Metropolis-Hastings, 430
Minkowski distance, 368
model-based clustering, 380
mom, 223
moment index, 530

moments, 22
Monte Carlo integration, 427
Monte Carlo methods, 191
  Type I error, 210
  Type II error, 211
multiplication rule, 18
mutually exclusive, 14

## N

Nadaraya-Watson estimator, 404
nearest neighbor event-event distance, 478, 486
nearest neighbor point-event distance, 478, 486
normal distribution, 31
  random number generation, 89
normal distribution, multivariate, 41
  random number generation, 96
**normcdf**, 32, 198, 200
**norminv**, 124, 196, 203, 212, 332
**normpdf**, 290
**normplot**, 123, 207
**normspec**, 32
notation, 5
null hypothesis, 192

## O

**ones**, 517
order of model, 233
order statistic, 69
outliers, 133

## P

parallel coordinates, 162, 168
parameter space, 60
parametric bootstrap, 216
**patch**, 151
pattern recognition, 317
  classification rate, 333
  cross-validation, 335, 339
  false alarms, 330
  hypothesis testing, 329
  independent test sample, 333
  likelihood ratio, 329
**pcolor**, 138
**pdist**, 369
**peaks**, 139
**pie**, 183
**plot**, 136, 522
**plot3**, 145, 522
**plotmatrix**, 148
points, spatial, 468
**poisscdf**, 28
Poisson distribution, 26, 448
  random number generation, 102
Poisson process, cluster, 499
Poisson process, homogeneous, 495
Poissonness plot, 126
**polyfit**, 121, 234, 388
**polyval**, 236
population, 52
posterior probability, 301, 320, 426
power, 197
prediction error, 236
predictor variable, 233
Priestley-Chao estimator, 404
principal component analysis, 168
prior probability, 320, 426
probability, 14
  axioms, 17
  conditional probability, 17
  distribution, 14
  posterior, 19
  prior, 19
probability density function, 14
probability mass function, 14
product kernel, 285
projection pursuit, 168, 186
  notation, 170
proposal distribution, 431
*p*-value, 199

## Q

**quad**, 48
**quadl**, 48
quadrats, 475
quantile plots, 119
  binomialness plot, 129
  normal probability plot, 123
  Poissonness plot, 126
  probability plot, 123

q-q plot, **119**, **387**
quantiles, **70**, **195**
quartiles, **70**

## R

**rand**, **80**
**randn**, **89**
random number generation
acceptance-rejection method, **85**
discrete uniform distribution, **104**
from density estimates, **306**
inverse transform method, **82**
variates on a sphere, **99**
random sample, **52**
random variable, **13**
continuous, **13**
discrete, **13**
random-walk Metropolis, **434**
**randperm**, **107**
**randtool**, **110**
receiver operating characteristic curve, **337**
regression, **385**
regression trees, **407**
cross-validation, **416**
error complexity measure, **411**
growing a tree, **410**
MATLAB code, **544**
notation, **408**
pruning a tree, **411**
selecting a tree, **412**
regression, parametric, **233**, **385**
regularity, spatial, **469**
rejection region, **194**
relative frequency method, **14**
residual dependence plot, **388**
residual squared error, **234**
response variable, **233**
**rotate3d**, **145**, **477**

## S

sample, **52**
sample mean, **53**
sample moments, **54**
sample space, **12**
sample variance, **54**
sampling distribution, **58**, **204**
**save**, **514**
**scatter**, **136**
**scatter3**, **145**
scatterplot, **135**, **186**, **231**
trivariate, **145**
scatterplot matrix, **148**
sequential spatial inhibition, **501**
shading, **138**, **189**
significance level, **195**
simple random sample, **52**
single linkage, **370**
skewness, **23**, **54**, **217**
skewness, quartile coefficient of, **77**
**slice**, **149**
smoothing, **390**
kernel methods, **401**
spatial point patterns, **466**
spatial statistics, **465**
splines, **419**
**squareform**, **369**
**stairs**, **183**, **279**
standard error, **62**
star plots, **155**
stationary point process, **468**
statistic, **53**
**stem**, **183**
stem-and-leaf, **117**, **185**
stratified random sampling, **53**
Strauss process, **504**
structure removal, **174**
Sturges' rule, **264**
**subplot**, **523**
supervised learning, **318**
**surf**, **138**
surface plots, **138**

## T

target distribution, **430**
**trapz**, **272**, **314**
two-tail test, **195**
Type I error, **195**
Type II error, **195**

## U

**unidrnd**, 89, 217
**unifcdf**, 29
uniform distribution, 29, 431
  random number generation, 79
unsupervised learning, 319
upper tail test, 195

## V

variance, 21

## W

Ward's linkage, 371
**weibcdf**, 40
**weibplot**, 123
Weibull distribution, 38
Workspace Browser, 512

## Z

**zeros**, 517