Computational applications in stochastic operations research

William H. Kaczynski
College of William & Mary - Arts & Sciences

Computational Applications in Stochastic Operations Research

William H. Kaczynski

Alvin, Texas

Bachelor of Science, United States Military Academy, 1992
Master of Science, Georgia Institute of Technology, 2002

A Dissertation presented to the Graduate Faculty
of the College of William & Mary in Candidacy for the Degree of
Doctor of Philosophy

Department of Applied Sciences

The College of William & Mary
August, 2009

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment
of
the requirements for the Degree of

Doctor of Philosophy

[illegible]

William H. Kaczynski

Approved by the Committee, June 2009

[illegible]

Committee Chair
Dr. Lawrence M. Leemis

[illegible]

Dr. John H. Drew

[illegible]

Dr. Rex K. Kincaid

[illegible]

Dr. Gregory D. Smith

[illegible]

Dr. Christopher A. Del Negro

ABSTRACT PAGE

Several computational applications in stochastic operations research are presented, where, for each application, a computational engine is used to achieve results that are otherwise overly tedious by hand calculations, or in some cases mathematically intractable. Algorithms and code are developed and implemented with specific emphasis placed on achieving exact results and substantiated via Monte Carlo simulation. The code for each application is provided in the software language utilized and algorithms are available for coding in another environment. The topics include univariate and bivariate nonparametric random variate generation using a piecewise-linear cumulative distribution, deriving exact statistical process control chart constants for non-normal sampling, testing probability distribution conformance to Benford's law, and transient analysis of $M/M/s$ queueing systems. The nonparametric random variate generation chapters provide the modeler with a method of generating univariate and bivariate samples when only observed data is available. The method is completely nonparametric and is capable of mimicking multimodal joint distributions. The algorithm is "black-box," where no decisions are required from the modeler in generating variates for simulation. The statistical process control chart constant chapter develops constants for select non-normal distributions, and provides tabulated results for researchers who have identified a given process as non-normal. The constants derived are bias correction factors for the sample range and sample standard deviation. The Benford conformance testing chapter offers the Kolmogorov–Smirnov test as an alternative to the standard chi-square goodness-of-fit test when testing whether leading digits of a data set are distributed according to Benford's law. The alternative test has the advantage of being an exact test for all sample sizes, removing the usual sample size restriction involved with the chi-square goodness-of-fit test. The transient queueing analysis chapter develops and automates the construction of the sojourn time distribution for the nth customer in an $M/M/s$ queue with k customers initially present at time 0 ($k \geq 0$) without the usual limit on traffic intensity, $\rho < 1$, providing an avenue to conduct transient analysis on various measures of performance for a given initial number of customers in the system. It also develops and automates the construction of the sojourn time joint probability distribution function for pairs of customers, allowing the calculation of the exact covariance between customer sojourn times.

Contents

Acknowledgements

I am extremely thankful to all those who provided so much of their time and assistance helping me to produce this dissertation as well as the research conducted. Many thanks to Professors Christopher Del Negro, John Drew, Rex Kincaid, and Greg Smith for being a part of the dissertation committee and providing meaningful assistance with the format, content, and overall dissertation product. I would also like to thank those who helped with several of the topics included in the dissertation, Professor John Drew, Professor Nick Loehr, and Jackie McQueston. I would especially like to recognize the tremendous efforts of my dissertation advisor, Professor Larry Leemis, who always managed to make himself available and was relentlessly positive, patient, insightful, and supportive of the research. I am forever indebted to my wife, Natalie, and my children, Madison, and Haley, for their understanding and patience during this time.

I would also like to acknowledge others who helped along the way. Michael Lewis assisted in formulating and verifying solutions of the nonlinear optimization program in Chapter 2. Barry Nelson assisted in applying empirical likelihood theory to weighting data values in Chapter 2. In Chapter 3 Jeff Robinson, from the General Motors Research and Development Center, provided the warranty data and Bruce Schmeiser described the two-dimensional empirical cumulative distribution function. Donald Wheeler was extremely generous in providing insight to normal and non-normal sampling for statistical process control chart constants in Chapter 4. In Chapter 6 Bert Zwart described the conditional discrete-event model displayed in Figure 6.6, Sheldon Ross offered assistance in verifying the covariance and correlation calculations, and Sid Lawrence, Barry Nelson, Raghu Pasupathy and Bruce Schmeiser assisted in deriving the sojourn time distributions.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Stochastic Operations Research

Operations research, as defined by Winston (2004), is "simply a scientific approach to decision making that seeks to best design and operate a system, usually under conditions requiring the allocation of scarce resources." Stochastic operations research is a subset of operations research in which the system of interest operates in the presence of some type of randomness. This science is almost always interdisciplinary in that several tools may be used simultaneously to achieve a desired result. Operations researchers use tools such as mathematical modeling, statistics, optimization, probability theory, queueing theory, and simulation. Using such a tool-kit to solve a problem often requires significant computing power, off-the-shelf or custom software, software-specific knowledge, and system-specific experience. The use of stochastic operations research is growing more common in business and industry, especially in the areas of revenue management and evaluating best practices.

The availability of off-the-shelf software provides increasing opportunities to apply known theory to real problems, and the availability of cheap computing has revolutionized applications in operations research. Current challenges involve researchers' abilities to not only tailor computer languages and software to address specific problems, but also to interpret output and state meaningful conclusions. The intent of this dissertation is to present several computational applications in stochastic operations research. The applications presented use a variety of computational engines to achieve exact results to known and new problems, generate random variates for simulation, test data for conformance to a known probability distribution, and verify results via simulation. Some of these exact results are novel in theory and application, others support previously known, but only simulated results, extending the current literature. In each case, the code used for a specific problem is available in the appendices. Where appropriate, segments of code are provided in the main text for illustration. Additionally, to substantiate exact results, Monte Carlo and discrete-event simulation code is also provided where necessary. Of particular interest are the exact symbolic results provided throughout the document, which highlight the ability of computer algebra systems to efficiently compute in symbolic form. The software utilized in the dissertation does not suggest a preference; there are many alternatives that could be used as appropriate substitutes.

1.2 Software

The thread linking the applications that appear in this document is a computational engine. Each chapter uses at least one software program; a few chapters use several. The only computer algebra system used is Maple. The reason I chose to use Maple involves the use of A Probability Programming Language (APPL) written by Glen et al. (2001). APPL is a compilation of Maple statements packaged conveniently to manipulate random variables with arbitrary distributions. APPL, and subsequently Maple, are used in Chapters 4, 5, and 6.

S-Plus and R are statistical and graphical software platforms. Although they have extensive capability in statistical analysis and computing, they are not capable of manipulating symbols, thus they are not considered computer algebra systems. S-

Plus and R are primarily used in this research for complicated algorithm processing, discrete-event simulation, Monte Carlo simulation, and graphics. Their use appears in Chapters 2, 3, 4, 5, and 6. When possible, algorithms written in S-Plus and R manipulate matrices and vectors, significantly enhancing algorithm speed. It should be noted however, that the main purpose of this work is not a computer science-focused work on computational complexity, therefore the author has occasionally chosen clarity over speed in designing algorithms.

Less prevalent, Microsoft Excel, MATLAB, and C also appear in the dissertation. Excel and MATLAB both possess solvers that are used in Chapter 2 as computational engines to solve a nonlinear optimization problem. C is used in Chapters 5 and 6, primarily for its speed as a compiled language. There are certainly many other software packages that could accomplish the same tasks of those listed above. However, regardless of the package used, my focus is on implementing some type of computational device for an application that is otherwise overly tedious or even intractable.

1.3 Literature Review

Except for Chapters 2 and 3, which are linked, each chapter of this dissertation differs substantially in content. Reviewing the literature in a single location of the dissertation would break a logical flow in the document. Therefore, the first section of each chapter provides an appropriate literature review for the selected chapter topic. The review for each chapter relied on recommendations from scholars in the appropriate fields, and articles in the fields. Each chapter has been submitted for journal publication.

1.4 Outline of the Dissertation

This section provides a short outline of the topics that follow in each chapter, and why the use of a computational engine is included. The problem is introduced and a brief overview of the chapter follows. The topics are merely introduced; for a detailed treatment please refer to the associated chapter.

Chapter 2 addresses univariate nonparametric variate generation. The standard approach to solving the interpolation problem for a trace-driven simulation involving a continuous random variable is to construct a piecewise-linear cumulative distribution function (CDF) that fills in the gaps between the data values. This approach overcomes the interpolation problem associated with simply resampling the data. Some probabilistic properties of the piecewise-linear estimator are derived, and two extensions to the standard approach (matching moments and weighted values) are presented, along with associated random variate generation algorithms. The algorithm is a nonparametric blackbox variate generator requiring only observed data from the modeler. The algorithm is implemented in S-Plus/R, where the setup portion matches the first two moments of the estimator to the first two moments of the data, then the execution portion generates a single variate from the piecewise-linear CDF created from the adjusted observed data.

Chapter 3 contains an extension of the univariate case of nonparametric random variate generation using a piecewise-linear cumulative distribution function to the bivariate case. The method is also a blackbox variate generation technique requiring only data pairs from the modeler. The technique avoids the time consuming and often arbitrary process of density estimation along with the potential error associated with estimation. It effectively captures marginal distributions with multiple modes. The algorithm implemented in S-Plus/R uses the convex hull of the observed data as a preliminary support, then generates the first element of the two-dimensional random vector via inversion of the marginal piecewise-linear CDF, and the second element from a conditional weighted piecewise-linear CDF created from selected values of the second variable. This procedure is especially tedious to implement by hand since a new conditional weighted piecewise-linear CDF is created for each bivariate pair generated. This proposed method is compared to the leading nonparametric method, kernel density estimation, and examples are provided with detailed results on the performance of each method.

In Chapter 4, expressions for statistical process control chart constants are developed and computed for non-normal sampling. Statistical process control chart constants are bias correction factors used to establish three-sigma limits that are used to identify assignable variation in a system. These constants allow engineers who monitor processes via periodic sampling to identify system-specific occurrences outside what would be considered normal operating bounds. Problems are potentially identified in near real-time as opposed to, for example, producing an entire lot of a component that is outside of specifications. These constants have only been tabulated for normal sampling (i.e., the measure of interest is normally distributed). The chapter uses APPL and Maple to obtain exact process control chart constants for both the normal distribution and select non-normal distributions. For populations clearly exhibiting non-normal distribution behavior, non-normal control chart constants are more appropriate.

Chapter 5 develops the use of the Kolmogorov–Smirnov (KS) test as an alternative to the chi-square goodness-of-fit test for assessing whether data conforms to Benford's law. Both approaches are compared for select distributions and results concerning the power of each test are provided as a means for selection. Benford's law states that in data sets satisfying certain conditions the leading digit X is distributed as

$$f_X(x) = P(X = x) = \log_{10}(1 + 1/x), \qquad x = 1, 2, \ldots, 9.$$

Therefore, the digit 1 appears most often and each subsequent digit appears less frequently with the digit 9 appearing the least often. A Monte Carlo simulation is implemented in S-Plus/R to compare the tests. Applications of Benford's law are becoming more popular to identify financial fraud in business and voting fraud.

Chapter 6 contains derivations of the exact distribution of the nth customer sojourn time in an $M/M/s$ queue with k customers initially present. Algorithms for computing the covariance between sojourn times for an $M/M/1$ queue with k customers initially present are also developed. Computer code is provided in the Maple environment for practical application of transient queue analysis for many system measures of performance without regard to traffic intensity (i.e., the system may be unstable with traffic intensity greater than one). The traffic intensity is defined as the customer arrival rate divided by the service rate. In steady-state queueing analysis the traffic intensity is restriced to a value less than one. However, many queueing systems of interest never achieve steady-state. The computational demand in this chapter is extensive. Without the use of APPL and Maple, results for systems larger than three customers are unrealistic. However, using the computational engine provides exact numeric and symbolic results.

The dissertation concludes with Chapter 7, where the results are briefly reviewed and areas of future work are discussed.

Chapter 2

Univariate Nonparametric Random Variate Generation

2.1 Introduction

Simulation practitioners often advocate a "trace-driven" approach to input modeling, in which data values are sampled with equal probability. In the univariate case, this approach is equivalent to generating variates from the empirical cumulative distribution function (CDF)

$$\hat{F}(x) = \frac{N(x)}{n} \qquad -\infty < x < \infty,$$

where n is the sample size, $N(x)$ is the number of data values less than or equal to x, and $x_1, x_2, \ldots, x_n$ denote the data values. We limit the discussion here to the case of raw data, rather than grouped data.

The advantages to the trace-driven approach are that (a) it avoids any error that might be introduced by fitting the data with an approximate parametric model, and (b) the sampling technique is identical to bootstrapping (Efron and Tibshirani, 1993) and, hence, has well-established statistical properties.

The disadvantages to the trace-driven approach are that (*a*) no random variate can be generated between the data values, known as the interpolation problem, and (*b*) no random variate can be generated that is smaller than the smallest data value or larger than the largest data value, known as the extrapolation problem.

A standard technique for overcoming the interpolation problem is to replace the empirical CDF with a CDF which is piecewise linear between the data values (Banks, Carson, Nelson, and Nicol, 2001, pages 296–300; Law, 2007, pages 309–310 and page 458; Leemis and Park, 2006, pages 409–411). Since the $n-1$ gaps between the data values should assume equal weighting, the piecewise-linear CDF has the form

$$\tilde{F}(x) = \begin{cases} 0 & x < x_{(1)} \\ \dfrac{i-1}{n-1} + \dfrac{x - x_{(i)}}{(n-1)(x_{(i+1)} - x_{(i)})} & x_{(i)} \le x < x_{(i+1)};\, i = 1, 2, \ldots, n-1 \\ 1 & x \ge x_{(n)}, \end{cases}$$

where $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$ are the order statistics, i.e., the data values sorted into ascending order. This CDF passes through the points

$$(x_{(1)}, 0), \left(x_{(2)}, \frac{1}{n-1}\right), \left(x_{(3)}, \frac{2}{n-1}\right), \ldots, (x_{(n)}, 1),$$

which we refer to as "knot points."

Example 1. Consider the univariate data set of $n = 6$ observations:

$$1 \quad 2 \quad 5 \quad 7 \quad 8 \quad 9.$$

We assume that these data values are drawn from a continuous population. The empirical CDF and piecewise-linear CDF are shown in Figure 2.1. The piecewise-linear CDF strikes the risers of the empirical CDF; the first intersection occurs $1/5$ of the way up the riser at $x = 2$ and the second intersection occurs $2/5$ of the way up the riser at $x = 5$. This pattern continues until the piecewise-linear CDF strikes the top of the last riser at $x = 9$.

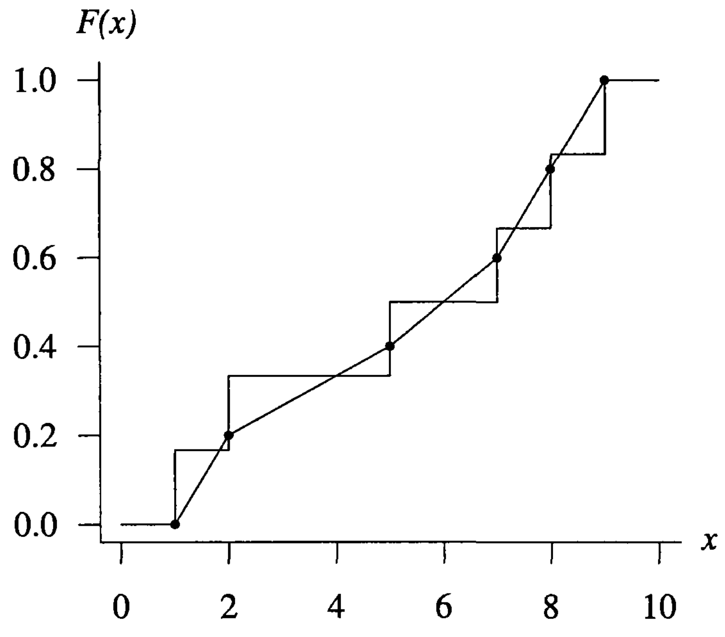


Figure 2.1: Empirical and piecewise-linear CDFs.

The probability density function (PDF) associated with the piecewise-linear CDF is constant between the data values:

$$\tilde{f}(x) = \frac{1}{(n-1)(x_{(i+1)} - x_{(i)})} \qquad x_{(i)} \le x < x_{(i+1)}; i = 1, 2, \ldots, n-1.$$

The mean of this distribution is

$$\begin{aligned}
E[X] &= \int_{x_{(1)}}^{x_{(n)}} x\tilde{f}(x)\,dx \\
&= \sum_{i=1}^{n-1} \int_{x_{(i)}}^{x_{(i+1)}} \frac{x}{(n-1)(x_{(i+1)} - x_{(i)})}\,dx \\
&= \sum_{i=1}^{n-1} \frac{x_{(i+1)}^2 - x_{(i)}^2}{2(n-1)(x_{(i+1)} - x_{(i)})} \\
&= \sum_{i=1}^{n-1} \frac{x_{(i)} + x_{(i+1)}}{2(n-1)} \\
&= \frac{x_{(1)} + 2x_{(2)} + 2x_{(3)} + \cdots + 2x_{(n-1)} + x_{(n)}}{2(n-1)}.
\end{aligned}$$

This weighted average of the data values places less weight on the extreme values, and equals $\bar{x}$, the sample mean of the data values, in only rare cases (e.g., a symmetric data set). The value of $E[X]$ approaches the sample mean $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ in the limit as $n \to \infty$. (The match between the coefficients in the expression for $E[X]$ and the coefficients in the trapezoidal rule is discussed in Appendix A.) Likewise, the second moment is

$$\begin{aligned}
E[X^2] &= \int_{x_{(1)}}^{x_{(n)}} x^2\tilde{f}(x)\,dx \\
&= \sum_{i=1}^{n-1} \int_{x_{(i)}}^{x_{(i+1)}} \frac{x^2}{(n-1)(x_{(i+1)} - x_{(i)})}\,dx \\
&= \sum_{i=1}^{n-1} \frac{x_{(i+1)}^3 - x_{(i)}^3}{3(n-1)(x_{(i+1)} - x_{(i)})} \\
&= \sum_{i=1}^{n-1} \frac{x_{(i)}^2 + x_{(i)}x_{(i+1)} + x_{(i+1)}^2}{3(n-1)}.
\end{aligned}$$

The variance of the distribution can be computed as $\sigma^2 = E[X^2] - E[X]^2$. For the data set from Example 1, the mean and variance of the piecewise-linear estimate are $E\,[X] = 16/3$ and $\text{Var}\,[X] = 71/9$.

Random variates can be generated efficiently by inverting the piecewise-linear CDF. Given $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$ and a random number generator, an $O(1)$ variate generation algorithm is:

```
generate $U \sim U(0,1)$
$i \leftarrow \lceil (n-1)U \rceil$
return $\left(x_{(i)} + ((n-1)\,U - (i-1))\left(x_{(i+1)} - x_{(i)}\right)\right)$
```

The index i, which assumes one of the integers $1, 2, \ldots, n-1$ with equal likelihood, determines which linear segment to invert. Although this O(1) algorithm is synchronized, monotone, and fast, there are three potential weaknesses that are described in the paragraphs below.

One potential weakness that arises with the piecewise-linear CDF $\tilde{F}(x)$ occurs when there are tied values in the data set. These tied values result in a discontinuity in $\tilde{F}(x)$. More specifically, when there are d tied values at $x_{(i)}$, there will be a discontinuity of height $d/(n-1)$ at $x_{(i)}$. The associated random variable is mixed (i.e., part discrete and part continuous), and the random variate generation algorithm will generate $x_{(i)}$ with probability $d/(n-1)$. If the modeler requires an absolutely continuous distribution, then it might be reasonable to use the midpoint of the discontinuity at $\tilde{F}(x_{(i)})$ as the knot point for the modified CDF. The variate generation algorithm would need to be modified appropriately.

A second serious weakness of the piecewise-linear approach is that data values that are close together (a common occurrence) lead to high peaks in the estimated density and an associated clustering of random variates near these particular data values. Two ways to overcome this weakness are to (*a*) use kernel density estimation, and (*b*) use the piecewise-linear approach on order statistics selected by discarding those with, for example, even indices. The pros and cons on these two alternative methods are addressed later in the chapter.

A third weakness is the extrapolation problem. Due to the finite end points of the piecewise-linear CDF, generating a variate below the first order statistic, $x_{(1)}$, and above the last order statistic of the sample, $x_{(n)}$, is impossible. Bratley, Fox, and Schrage (1987) offer Marsaglia's tail algorithm as an elegant way to generate from the tail of a distribution. This approach proves useful in extending possible variate generation beyond just the sample range of a data set.

In this chapter we present two alternatives that overcome these weaknesses. The alternatives to the piecewise-linear CDF are nonparametric, thus avoiding potential error associated with a parametric model. They also allow some extrapolation below the minimum and maximum data values by stretching and translating observed data values such that the estimator's mean and variance match the sample mean and variance. Chapter 2.2 develops these variants in detail and Chapter 2.3 compares resulting estimators with estimates based on kernel density estimation.

2.2 Moment Matching and Weighted Observations

We consider two variations on the piecewise-linear CDF as a probabilistic model for a data set drawn from a continuous population. The first variation adjusts the knot points in the piecewise-linear CDF so that its first and second moments match those from the data set. The second variation makes adjustments to the piecewise-linear CDF by allowing different weights for each of the data values.

2.2.1 Matching Moments

Occasions might arise when a modeler would like to (a) maintain the piecewise-linear nature of the CDF, (b) maintain the heights of the knot points at 0, $\frac{1}{n-1}, \frac{2}{n-1}, \ldots, 1$ (which implies fast variate generation), and (c) match the mean and variance of the piecewise-linear CDF to the sample mean and sample variance of the observations. This can only be achieved by adjusting the horizontal values of the knot points. We begin the development of this process with a simple example.

Example 2. Consider a data set consisting of just $n = 2$ observations: 0

and 1. This data set has sample mean $\bar{x} = \frac{1}{2}$ and unbiased sample variance $s^2 = \frac{1}{2}$. The piecewise-linear CDF for this data set is that associated with the $U(0,1)$ distribution, which has mean $\mu = \frac{1}{2}$ and variance $\sigma^2 = \frac{1}{12}$. The reduction in the variance associated with the piecewise-linear CDF is significant in this case because of the small sample size. One way to match variances is to shift the smaller data value to the left and shift the larger data value to the right by an equal amount δ for the piecewise-linear CDF. The appropriate shift δ satisfies

$$\frac{\left((1+\delta)-(0-\delta)\right)^2}{12} = \frac{1}{2}.$$

There are two roots to this quadratic equation. The positive root increases the larger data value and decreases the smaller data value. The negative root decreases the larger data value and increases the smaller data value by a large enough value so that their roles are reversed. For a symmetric data set like this one, either root will produce the same knot points. Since most data sets are not symmetric, we always choose the positive root, which is $\delta = (\sqrt{6}-1)/2 \cong 0.7247$ in this case. Thus a piecewise-linear CDF with knot points

$$(-0.7247, 0), (1.7247, 1)$$

has a variance which matches the variance of the original data values. (The means also happen to match in this case although this will not be true in general.)

The expansion of the support of the piecewise-linear cumulative distribution function beyond the outermost data values, as illustrated in the previous example, may not be appropriate for all modeling situations. If the data values collected are service times in a queuing model, for instance, spreading the observations might result in a support that includes negative service times. For the occasions when matching means and variances is appropriate, we derive the values of the knot points below. This derivation will maintain the ratios of the gaps between the data values so that their spreading is accomplished in the same way a bellows is spread on an accordion. We stretch the data to match variances first, then shift the data to match the means.

Let $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$ be the ordered raw data values as before and let

$$g_i = x_{(i+1)} - x_{(i)}$$

for $i = 1, 2, \ldots, n-1$ be the i^{th} gap between the observations. Let

$$g_i' = \frac{g_i}{\sum_{j=1}^{n-1} g_j} = \frac{g_i}{x_{(n)} - x_{(1)}}$$

for $i = 1, 2, \ldots, n-1$ be the normalized gap values. If $x_{(1)}$ is shifted to $x'_{(1)} = x_{(1)} - \delta$ and $x_{(n)}$ is shifted to $x'_{(n)} = x_{(n)} + \delta$, the width of the support of the adjusted piecewise-linear CDF is

$$w = x_{(n)} - x_{(1)} + 2\delta.$$

To maintain the ratios of the normalized gap values, the adjusted data values are

$$x'_{(i)} = x_{(1)} - \delta + w \sum_{j=1}^{i-1} g_j'$$

for $i = 1, 2, \ldots, n$. The root finding problem now reduces to finding the value δ such that the unbiased sample variance of the original data values $x_1, x_2, \ldots, x_n$ matches the variance of the piecewise-linear CDF associated with the adjusted data values.

Once the variances have been matched, the means are easily matched by shifting each adjusted data value

$$x''_{(i)} = x'_{(i)} - \left[\frac{x'_{(i)} + 2x'_{(2)} + \cdots + 2x'_{(n-1)} + x'_{(n)}}{2(n-1)} - \bar{x}\right]$$

for $i = 1, 2, \ldots, n$. So finally, the knot points of the piecewise-linear CDF that matches first and second moments with the data are

$$\left(x''_{(1)}, 0\right), \left(x''_{(2)}, \frac{1}{n-1}\right), \left(x''_{(3)}, \frac{2}{n-1}\right), \ldots, \left(x''_{(n)}, 1\right).$$

Random variate generation via inversion is performed by the algorithm given in the introduction using the $x''_{(i)}$. Since the differences between the heights of adjacent knot points is constant, variate generation is fast. The stretching and shifting partially solves the extrapolation problem by allowing random variates to be generated outside of the range of the data values. Additionally, in the limit as $n \to \infty$, the sample variance s^2 approaches the population variance σ^2. Therefore, with increasing n, the value of δ is decreasing and as $n \to \infty$, $\delta \to 0$. Additionally, δ must exist since it is well known that the variance of the piecewise-linear estimator is always less than the variance of the sample data, and therefore, by construction, there exists $\delta > 0$ such that the adjusted data points equate the variance of the piecewise linear estimator and the sample variance of the data.

Example 3. Consider again the $n = 6$ data values

$$1 \quad 2 \quad 5 \quad 7 \quad 8 \quad 9.$$

Find the piecewise-linear CDF knot values with matching means and variances. In order to match both the mean and variance, we first match the variances by stretching the data, then apply a shift that matches the means. For the ordered data values

$$x_{(1)} = 1, x_{(2)} = 2, x_{(3)} = 5, x_{(4)} = 7, x_{(5)} = 8, x_{(6)} = 9$$

with gaps

$$g_1 = 1, g_2 = 3, g_3 = 2, g_4 = 1, g_5 = 1$$

and associated normalized gaps

$$g_1' = \frac{1}{8}, g_2' = \frac{3}{8}, g_3' = \frac{2}{8}, g_4' = \frac{1}{8}, g_5' = \frac{1}{8}$$

the adjusted data values are

$$\begin{aligned}
x_{(1)}' &= 1 - \delta \\
x_{(2)}' &= 1 - \delta + (8 + 2\delta)\frac{1}{8} = 2 - \frac{3\delta}{4} \\
x_{(3)}' &= 1 - \delta + (8 + 2\delta)\frac{4}{8} = 5 \\
x_{(4)}' &= 1 - \delta + (8 + 2\delta)\frac{6}{8} = 7 + \frac{\delta}{2} \\
x_{(5)}' &= 1 - \delta + (8 + 2\delta)\frac{7}{8} = 8 + \frac{3\delta}{4} \\
x_{(6)}' &= 1 - \delta + (8 + 2\delta) = 9 + \delta.
\end{aligned}$$

The sample mean of the data is

$$\bar{x} = \frac{1 + 2 + 5 + 7 + 8 + 9}{6} = \frac{16}{3}$$

and the unbiased sample variance of the data is

$$s^2 = \frac{1}{5}\left[\left(1 - \frac{16}{3}\right)^2 + \left(2 - \frac{16}{3}\right)^2 + \cdots + \left(8 - \frac{16}{3}\right)^2 + \left(9 - \frac{16}{3}\right)^2\right] = \frac{32}{3}.$$

When the adjusted data values are used as arguments in the formula for the variance of the piecewise-linear CDF, the value of δ must satisfy the quadratic equation

$$\left[\frac{(1-\delta)^2 + (1-\delta)(2 - 3\delta/4) + 2(2 - 3\delta/4)^2 + \cdots + (8 + 3\delta/4)(9 + \delta) + (9 + \delta)^2}{(3)(5)}\right]$$
$$- \left[\frac{(1-\delta) + 2(2 - 3\delta/4) + \cdots + 2(8 + 3\delta/4) + (9 + \delta)}{(2)(5)}\right]^2 = \frac{32}{3}$$

which reduces to

$$\frac{518}{75}+\frac{259\delta}{75}+\frac{259\delta^2}{600}=\frac{32}{3}.$$

This quadratic equation has positive root

$$\delta=-4+\frac{80}{259}\sqrt{259}\cong 0.9710.$$

Selecting the negative root still matches the variance to that of the piecewise-linear CDF. However, selecting the negative root of the quadratic equation projects each of the original ordered data values about $(x_{(1)}+x_{(n)})/2$, which is only harmless for a symmetric data set. Finally, to match means,

$$\begin{aligned}
x''_{(1)} &= \frac{16}{3}-\frac{88}{259}\sqrt{259}\cong -0.1347\\
x''_{(2)} &= \frac{16}{3}-\frac{68}{259}\sqrt{259}\cong 1.1080\\
x''_{(3)} &= \frac{16}{3}-\frac{8}{259}\sqrt{259}\cong 4.8362\\
x''_{(4)} &= \frac{16}{3}+\frac{32}{259}\sqrt{259}\cong 7.3217\\
x''_{(5)} &= \frac{16}{3}+\frac{52}{259}\sqrt{259}\cong 8.5645\\
x''_{(6)} &= \frac{16}{3}+\frac{72}{259}\sqrt{259}\cong 9.8072
\end{aligned}$$

are the x-values associated with the knot points.

An algorithm for adjusting the data values so that the first two moments of the piecewise-linear model match those of the raw data is (indentation denotes nesting):

```
Input data values $x_1, x_2, \ldots, x_n$
$\bar{x} \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$
$s^2 \leftarrow \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2$
Sort the data values yielding $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$
$w \leftarrow x_{(n)} - x_{(1)} + 2\delta$
```

```
for $i \leftarrow 1$ to $n-1$
    $g_i \leftarrow x_{(i+1)} - x_{(i)}$
    $g'_i \leftarrow g_i/(x_{(n)} - x_{(1)})$
for $i \leftarrow 1$ to $n$
    $x'_{(i)} \leftarrow x_{(1)} - \delta + w\sum_{j=1}^{i-1} g'_j$
Find the positive root $\delta$ of the quadratic equation
```

$$\sum_{i=1}^{n-1} \frac{(x'_{(i)})^2 + x'_{(i)}x'_{(i+1)} + (x'_{(i+1)})^2}{3(n-1)} - \left[\frac{x'_{(1)} + 2\sum_{i=2}^{n-1} x'_{(i)} + x'_{(n)}}{2(n-1)}\right]^2 = s^2$$

```
for $i \leftarrow 1$ to $n$
```

$$x''_{(i)} \leftarrow x'_{(i)} - \left[\frac{x'_{(1)} + 2\sum_{i=2}^{n-1} x'_{(i)} + x'_{(n)}}{2(n-1)} - \bar{x}\right]$$

This piecewise-linear model associated with data values $x''_{(1)}, x''_{(2)}, \ldots, x''_{(n)}$ has a mean and variance that matches the mean and variance of the original data values. The only nontrivial step in this algorithm is solving the quadratic equation. This is easily done in a computer algebra system with its symbolic processing capabilities, but is more problematic for a standard algorithmic language. Appendix B contains an algorithm and associated S-Plus/R code for computing δ and $x''_{(1)}, x''_{(2)}, \ldots, x''_{(n)}$.

2.2.2 Weighted Data Values

One of the algorithms presented in Chapter 3 concerning the generation of bivariate observations, requires a variant of the univariate piecewise-linear CDF approach which allows for the data values to be weighted. Let $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$ be the sorted observations and $w_{(1)}, w_{(2)}, \ldots, w_{(n)}$, where $\sum_{i=1}^{n} w_{(i)} = 1$, be the corresponding positive-valued weights. Any estimated CDF should collapse to $\hat{F}(x)$ when $w_{(i)} = 1/n$, $i = 1, 2, \ldots, n$. Although there is no claim made to the uniqueness of the estimator presented here, one approach is to first draw the CDF associated with a discrete random variable X with support values $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$ and corresponding mass values $w_{(1)}, w_{(2)}, \ldots, w_{(n)}$. Points on each of the risers can be connected to form a piecewise-linear estimated CDF. The only question that remains is what the heights of these points should be. One reasonable approach is to place the first knot point at $(x_{(1)}, 0)$, the second knot point $\frac{1}{n-1}$ of the way up the second riser (which is associated with $x_{(2)}$), the third knot point $\frac{2}{n-1}$ of the way up the third riser (which is associated with $x_{(3)}$), and so on. Using this approach is equivalent to connecting the points

$$\left(x_{(1)}, 0\right), \left(x_{(2)}, w_{(1)} + \frac{w_{(2)}}{n-1}\right), \left(x_{(3)}, w_{(1)} + w_{(2)} + \frac{2w_{(3)}}{n-1}\right), \ldots, \left(x_{(n)}, 1\right)$$

to form the piecewise-linear CDF. Define

$$y_{(i)} = w_{(1)} + w_{(2)} + \cdots + w_{(i-1)} + \frac{(i-1)w_{(i)}}{n-1} \qquad i = 1, 2, \ldots, n,$$

as the height of each knot point. The piecewise-linear CDF for the weighted data values is

$$F^*(x) = \begin{cases} 0 & x < x_{(1)} \\ y_{(i)} + \dfrac{(y_{(i+1)} - y_{(i)})(x - x_{(i)})}{x_{(i+1)} - x_{(i)}} & x_{(i)} \le x < x_{(i+1)}; i = 1, 2, \ldots, n-1 \\ 1 & x \ge x_{(n)}. \end{cases}$$

This CDF reduces to $\tilde{F}(x)$ in the equal-weighting case when $w_{(i)} = 1/n$, for $i = 1, 2, \ldots, n$. Using the associated probability density function, it can be shown that $E[X]$ and $E[X^2]$ are

$$E[X] = \frac{1}{2} \sum_{i=1}^{n-1} \left(w_i + \frac{iw_{i+1} - (i-1)w_i}{n-1}\right) \left(x_{(i+1)} + x_{(i)}\right)$$

$$E[X^2] = \frac{1}{3} \sum_{i=1}^{n-1} \left(w_i + \frac{iw_{i+1} - (i-1)w_i}{n-1}\right) \left(x_{(i+1)}^2 + x_{(i+1)}x_{(i)} + x_{(i)}^2\right).$$

Using these results, we can calculate the variance of the weighted piecewise linear CDF via $\text{Var}[X] = E[X^2] - [E[X]]^2$.

To formulate an algorithm for variate generation, first sort the data, yielding the $x_{(i)}$ and $w_{(i)}$ values. Then, at the beginning of a simulation, calculate the $y_{(i)}$ values. The $O(n)$ algorithm for generating random variates given below also uses inversion.

```
Generate $U \sim U(0,1)$
$i \leftarrow 1$
while $(U > y_{(i+1)})$
    $i \leftarrow i+1$
return $\left(x_{(i)} + \left(U - y_{(i)}\right)\left(x_{(i+1)} - x_{(i)}\right) / \left(y_{(i+1)} - y_{(i)}\right)\right)$
```

As expected, this algorithm collapses to the equally-weighted algorithm given in Chapter 2.1 because $y_{(i)} = (i-1)/(n-1)$, for $i = 1, 2, \ldots, n$ in the equally-weighted case. This algorithm can easily be modified to a $O(\log n)$ algorithm by employing a binary search rather than the linear search presented here.

Occasions might arise in which the weights need to be calculated from data. Consider the previous example. The data values $1, 2, 5, 7, 8$, and 9 were stretched and translated so that the sample mean and variance matched the mean and variance of the piecewise-linear estimate. This resulted in the lowest data value $x_{(1)} = 1$ being shifted to $x''_{(1)} = -0.1347$. For certain types of data sets (e.g., service times), generating a negative service time might be unacceptable. So the only recourse for a modeler who wants to (a) keep the x-coordinates of the knot points at the data values and (b) match moments, is to adjust the weights $w_1, w_2, \ldots, w_n$ to values other than the usual equally-likely weights $1/n$. As seen earlier, the effect of moving from a data set to the piecewise-linear estimator is to decrease the variance. Thus adjusting the weights will place increased weight on the extreme values (and therefore less weight on the middle values) so as to increase the variance.

One problem that arises from this approach to matching moments is that there will typically not be a unique solution for the weights that will match moments. We therefore introduce the objective function

$$\frac{\prod_{i=1}^{n} w_i}{\prod_{i=1}^{n} 1/n}$$

from the empirical likelihood literature (Owen, 2001) to achieve a unique solution. Thus the optimization problem is nonlinear and is written with constraints as:

$$\begin{array}{ll}
\text{maximize} & n^n \prod_{i=1}^{n} w_i \\
\text{subject to} & \frac{1}{2}\sum_{i=1}^{n-1}\left(w_i + \frac{iw_{i+1}-(i-1)w_i}{n-1}\right)\left(x_{(i+1)}+x_{(i)}\right) = \bar{x} \\
& \frac{1}{3}\sum_{i=1}^{n-1}\left(w_i + \frac{iw_{i+1}-(i-1)w_i}{n-1}\right)\left(x_{(i+1)}^2+x_{(i+1)}x_{(i)}+x_{(i)}^2\right) \\
& \quad -\left(\frac{1}{2}\sum_{i=1}^{n-1}\left(w_i + \frac{iw_{i+1}-(i-1)w_i}{n-1}\right)\left(x_{(i+1)}+x_{(i)}\right)\right)^2 = s^2 \\
& \sum_{i=1}^{n} w_i = 1 \\
& w_i \geq 0.
\end{array}$$

This method is advantageous for certain types of positive data that might be close to zero, ensuring that negative x values are not created by stretching the data (e.g., positive service times). By choosing this method the x_i values are not affected.

Example 4. Consider the univariate data set of $n = 6$ observations:

$$1 \quad 2 \quad 5 \quad 7 \quad 8 \quad 9.$$

Just as in Example 1, we assume that these data values are drawn from a continuous population. The sample mean and sample variance of the data are $\overline{x} = 16/3$ and $s^2 = 32/3$. Find the corresponding weights, w_i, for $i = 1, 2, \ldots, 6$ that solve the above nonlinear program.

This problem was solved in Microsoft Excel and Matlab, yielding the optimal weights

$$w_1 = 0.3721, w_2 = 0.0519, w_3 = 0.0391,$$
$$w_4 = 0.0444, w_5 = 0.0761, w_6 = 0.4165.$$

These weights maximize the objective function and match the sample mean and variance of the data to the mean and variance of the weighted piecewise-linear CDF. The small sample size results in heavy weights being placed on the extreme values in order to match the moments.

Because this is a nonlinear optimization program, the solution achieved is quite sensitive to the solver chosen and starting point provided. As expected, as the number of observations n increases, the optimization problem becomes more difficult to solve. The next example uses a data set from survival analysis.

Example 5. Consider the univariate data set of $n = 23$ ball bearing failure times in millions of revolutions (Lieblein and Zelen, 1956):

$$17.88, 28.92, 33.00, 41.52, 42.12, 45.60, 48.48, 51.84, 51.96, 54.12, 55.56, 67.80,$$
$$68.64, 68.64, 68.88, 84.12, 93.12, 98.64, 105.12, 105.84, 127.92, 128.04, 173.40.$$

We assume that these data values are drawn from a continuous population. Find the corresponding weights, w_i for $i = 1, 2, \ldots, 23$ that solve the above nonlinear program.

This problem was again solved in Microsoft Excel and Matlab, yielding the optimal weights

$$w_1 = 0.0665,\ w_2 = 0.0552,\ \ldots,\ w_{22} = 0.0471,\ w_{23} = 0.0850.$$

Figure 2.2 shows the piecewise-linear CDF for this data and is overlaid with the optimal weighted piecewise-linear CDF matching the sample means and variances.

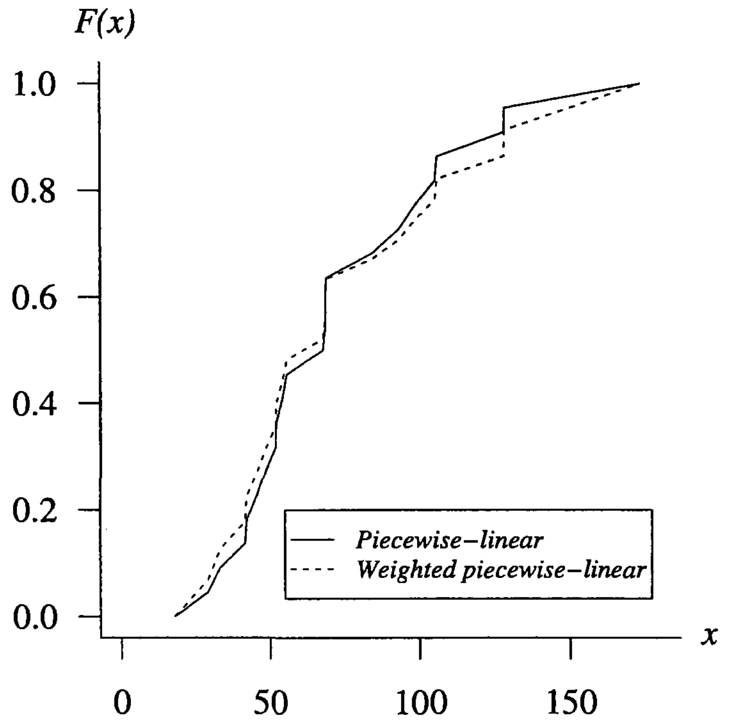


Figure 2.2: Piecewise-linear and optimal weighted piecewise-linear CDFs.

2.3 Comparing Estimates

Thus far, four estimates have been suggested for generating from a given continuous data set. They are (a) the piecewise-linear CDF, (b) the piecewise-linear CDF with a mean and variance matched to the data, (c) the weighted piecewise-linear CDF, and (d) the piecewise-linear CDF created by order statistics associated with discarding even indices. These methods all provide a means for variate generation via inversion, thus are fast, synchronized, and exact. Their main competitor in the literature is variate generation from an estimated density known as the kernel density. For a detailed discussion of this method, see Silverman (1986). To compare results for these estimates, a Monte Carlo simulation study was conducted in which estimates are created from six known candidate parametric distributions. These distributions were selected to adequately cover decreasing failure rate (DFR), increasing failure rate (IFR), increasing/decreasing failure rate (IFR/DFR), bathtub (BT), and upside-down bathtub (UBT) hazard functions. A sample was generated from each distribution, and the corresponding estimates were created. The metric developed for comparing the CDFs is

$$\frac{1}{nb}\sum_{j=1}^{b}\sum_{i=1}^{n}\left|F\left(x_{(i)}\right)-\ddot{F}_j\left(x_{(i)}\right)\right|,$$

where $F\left(x_{(i)}\right)$ is the CDF for the known population distribution at $x_{(i)}$, $\ddot{F}_j\left(x_{(i)}\right)$ is the corresponding jth CDF estimate at $x_{(i)}$ for one of the estimates listed below, n is the sample size, and b is the number of simulation replications. The average absolute errors for various sample sizes are given in Table 2.1, each for $b = 1,000,000$ replications. Common random numbers were used in the simulation to reduce the variability of the estimators. The results can be replicated using the `set.seed(123)` command. The smallest metric in each column is set in boldface type. The four estimators that are compared are:

- piecewise-linear estimator $\tilde{F}(x)$,
- moment matching piecewise-linear estimator $F^*(x)$,
- selected order statistic estimator $F^g(x)$,
- kernel estimator $F^k(x)$.

The selected order statistic estimator breaks up the clumping that occurs with random sampling by deleting every order statistic with an even index and using the piecewise-linear estimator on the remaining order statistics. This is why the sample sizes are chosen to be odd. The weighted piecewise-linear CDF method is not included in the study due to the CPU time required to solve multiple replications of the optimization problem. Two kernel functions were selected for the study: (a) the standard normal and (b) $U(-1,1)$. The bandwidth parameter used for each kernel density estimate is the optimal bandwidth parameter (Silverman, 1986) described by

$$b = \alpha(k) 1.364 \min(s, R/1.34) n^{-1/5}$$

where $\alpha(k) = 0.776$ for the Gaussian kernel, $\alpha(k) = 1.351$ for the uniform kernel, s is the sample standard deviation, and R is the sample range. The results for the uniform kernel density were not included because the estimates had gaps in their support. As expected, the kernel density estimate dominates for distributions with a pronounced mode. However, the arctangent, exponential, and bi-modal exponential power distributions are more accurately estimated by a piecewise-linear CDF. The matching moments estimator $\tilde{F}(x)$ for the exponential distribution deserves further explanation. When stretching values to match variances, negative values are possible, causing the excess error in the metric. We decided to leave this result as is in Table 2.1 with explanation for emphasis. In conclusion, though we boldface only one error value for each row of the table (except where ties occur), in many cases the average error differences between methods appear negligible.

$n = 9$	class	$\tilde{F}(x)$	$F^*(x)$	$F^g(x)$	$F^k(x)$
Uniform$(0, 1)$	IFR	0.112	0.105	0.110	**0.091**
Weibull$(1, 1/2)$	DFR	0.238	0.229	0.250	**0.213**
Exponential(1)	IFR/DFR	0.112	0.118	**0.110**	**0.110**
Weibull$(1, 2)$	IFR	0.112	0.099	0.110	**0.092**
Exponential Power$(1, 1/2)$	BT	0.158	0.169	**0.143**	0.170
Arctan$(1, 1)$	UBT	0.190	0.164	0.201	**0.161**
$n = 21$	class	$\tilde{F}(x)$	$F^*(x)$	$F^g(x)$	$F^k(x)$
Uniform$(0, 1)$	IFR	0.070	0.068	0.069	**0.061**
Weibull$(1, 1/2)$	DFR	0.219	0.216	0.222	**0.208**
Exponential(1)	IFR/DFR	0.070	0.094	**0.069**	0.088
Weibull$(1, 2)$	IFR	0.070	0.066	0.069	**0.062**
Exponential Power$(1, 1/2)$	BT	0.141	0.150	**0.134**	0.151
Arctan$(1, 1)$	UBT	0.164	**0.150**	0.168	0.151
$n = 45$	class	$\tilde{F}(x)$	$F^*(x)$	$F^g(x)$	$F^k(x)$
Uniform$(0, 1)$	IFR	0.047	0.046	0.047	**0.043**
Weibull$(1, 1/2)$	DFR	0.211	0.210	0.212	**0.206**
Exponential(1)	IFR/DFR	**0.047**	0.082	**0.047**	0.073
Weibull$(1, 2)$	IFR	0.047	0.046	0.047	**0.043**
Exponential Power$(1, 1/2)$	BT	0.136	0.142	**0.133**	0.141
Arctan$(1, 1)$	UBT	0.152	**0.135**	0.154	0.146
$n = 71$	class	$\tilde{F}(x)$	$F^*(x)$	$F^g(x)$	$F^k(x)$
Uniform$(0, 1)$	IFR	0.037	0.037	0.037	**0.035**
Weibull$(1, 1/2)$	DFR	0.208	0.208	0.209	**0.205**
Exponential(1)	IFR/DFR	**0.037**	0.076	**0.037**	0.067
Weibull$(1, 2)$	IFR	0.037	0.037	0.037	**0.035**
Exponential Power$(1, 1/2)$	BT	0.134	0.140	**0.132**	0.138
Arctan$(1, 1)$	UBT	0.148	**0.125**	0.149	0.144
$n = 101$	class	$\tilde{F}(x)$	$F^*(x)$	$F^g(x)$	$F^k(x)$
Uniform$(0, 1)$	IFR	0.031	0.031	0.031	**0.030**
Weibull$(1, 1/2)$	DFR	0.207	0.207	0.208	**0.205**
Exponential(1)	IFR/DFR	**0.031**	0.072	**0.031**	0.062
Weibull$(1, 2)$	IFR	0.031	0.031	0.031	**0.030**
Exponential Power$(1, 1/2)$	BT	0.134	0.138	**0.132**	0.137
Arctan$(1, 1)$	UBT	0.146	**0.118**	0.146	0.143

Table 2.1: Average absolute error.

2.4 Conclusions

The standard solution to the interpolation problem for Monte Carlo or discrete-event simulation uses a piecewise-linear CDF as a model. The variate generation algorithm is fast and trivial to implement. We have suggested two modifications to the original model: (a) stretching and shifting the original data values so that the mean and variance of the piecewise-linear CDF model matches the mean and variance of the sample values, and (b) a modification to the model and variate generation algorithm to account for weighted observations. Both of these modifications could prove to be useful in further work associated with the generation of bivariate samples.

We conclude with a summary of piecewise-linear and kernel density estimation pros and cons.

Piecewise-linear advantages:

- No decisions from the modeler; completely nonparametric
- Easily extended to match sample mean and variance
- Easily smoothed to minimize the effect of clustering of data values
- Extends to bivariate data without the assumptions and requirements demanded if using kernel density estimation

Kernel density estimation drawbacks:

- Arbitrary decisions left to the modeler
  - kernel density functional form
  - variance of kernel densities (smoothing parameter)
- Normal kernel density function implies an infinite left hand tail (obviously bad for certain data, i.e. service times)

- Inferior uniform density may
  - leave undesired gaps
  - extend to negative values

Piecewise-linear grouping drawbacks

- Grouping involves arbitrary decisions/parameters from the modeler
- Too much grouping may mask the shape of the distribution

While we recognize the approach presented in this chapter is "not ideal" in density estimation, our goal is not density estimation. The goal is nonparametric variate generation, thus density estimation can be considered as an unnecessary step. The method proposed is a turnkey operation, requiring only the observed data from the modeler. Additionally, the extension to the bivariate case is extremely desirable.

Chapter 3

Bivariate Nonparametric Random Variate Generation

3.1 Introduction

Parametric univariate random variate generation is a well-established methodology providing the modeler dozens of distribution choices having a variety of statistical properties (Banks, et al., 2001, Law, 2007, Leemis and Park, 2006). For parametric bivariate distributions, however, the number of distribution choices is much more limited. Additionally, the ability to generate observations from some bivariate distributions relies on the acceptance–rejection method, casting out the preferred inversion method. Recent literature in copula-based approaches indicates improvement in this area. Copula-based approaches have often been applied to finance and are becoming more prevalent in other areas such as actuarial science and hydrology. We did not consider these approaches as candidates for comparison because recent literature suggests that the method of model selection is not universally accepted (Genest and Rémillard, 2006). Additionally, this approach is a two-stage estimation process (1. marginals, 2. copula function). There is promising recent work in nonparametric copula-based approaches, overcoming the two-stage estimation issue. However, we do not compare the proposed algorithm to this work.

Kernel density estimation (KDE) is another popular method for density estimation. Hörmann and Leydold (2000) present algorithms that generate variates directly from a sample via KDE for both the univariate and bivariate cases. In their approach, resampling occurs from a multivariate normal distribution with a covariance matrix that matches that of the observed data. In the univariate case, Bratley, Fox, and Schrage (1987) and Law (2007) describe variate generation methods using the linear interpolation of the empirical distribution function. Generating variates from KDE offers the advantages (Devroye and Györfi, 1985, Devroye, 1986, Silverman, 1986) of simplicity and well-established theory of density estimation. However, KDE suffers from the arbitrary (but necessary) step of fine tuning a smoothing parameter as well as choosing the appropriate kernel function. Hörmann and Leydold (2000) also note that generating variates from KDE results in the "variance of the empirical distribution always being larger than the variance of the observed sample," and furthermore, since generating from KDE is not an inversion method, correlation induction for variance reduction is lost. Silverman (1986) presents an algorithm that corrects the KDE variance difference by forcing it to equal the sample variance.

Since the focus of this chapter is modeling bivariate dependencies in input data for simulation, we now review the literature in this area. In the parametric case, Devroye (1986) and Johnson (1987) devise strategies for generating from several multivariate distributions including the multi-normal and the multi-variate Johnson family. Wagner and Wilson (1995) develop techniques for the bivariate Bezier distribution. Taylor and Thompson (1986) formulate a semi-nonparametric method that comprises samples from a combination of a nearest neighbor technique and KDE. Matching moments occurs in some methods as an appropriate means for density estimation. Because the majority of these published methods assume a known population distribution, they are coupled with potentially unrealistic distribution properties such as the support, moments, etc. Additionally, many of these methods rely on the acceptance–rejection technique for variate generation, and thus are not synchronized. This loss in synchronization sacrifices the ability to implement variance reduction through the use of common random numbers, and carries the added expense of wasted $U(0,1)$'s. For all the literature reviewed, the two-dimensional random vectors can handle a single mode, and very few are capable of representing two-mode marginal distributions. We were unable to find a flexible family capable of greater than two modes, therefore generating variates according to some parametric family may not be possible for data with more than two modes.

In this chapter we intend to show that the proposed bivariate nonparametric random variate generation algorithm has three specific advantages over its primary competitor, KDE. The advantages are (1) no reliance on the selected kernel density function, (2) no reliance on the selected smoothing parameter, and (3) cannot produce unrealistic variates (i.e. negative values from a service time distribution).

The chapter is organized as follows. Section 3.2 first introduces a piecewise-linear CDF and explains how to sample from this CDF. It follows with a discussion of how to manipulate this estimator so that the first two moments of the estimator match the corresponding moments of the observed data. The section concludes with the proposed bivariate random variate generation algorithm, an applied example, and an interesting variant of the algorithm for selected data sets. Section 3.3 compares the proposed algorithm to KDE for bivariate data with unknown underlying bivariate densities, along with data generated from known bivariate densities. Where possible, the comparisons include visual representations, marginal means and variances, covariances, and squared error between the known and estimated CDFs. The last section summarizes the results.

3.2 Generating Variates From Bivariate Data

One obvious and simple technique for generating variates from a data set

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

is to sample from the empirical CDF

$$\hat{F}(x, y) = \frac{1}{n} I(x, y),$$

where $I(x, y)$ is a function that counts the number of (x_i, y_i) pairs in the data set satisfying $x_i \le x$ and $y_i \le y$ (i.e., $\hat{F}(x, y)$ is the fraction of the data pairs falling to the southwest of (x, y)). An algorithm for generating from the empirical CDF is equivalent to sampling with replacement from the data pairs (x_i, y_i):

```
1. generate $U \sim U(0, 1)$
2. $I \leftarrow \lceil nU \rceil$
3. return $(x_I, y_I)$
```

This random variate generation technique is fast and conceptually straightforward. The drawback with this method is that the random variates are limited to the data pairs—which is particularly problematic for a small sample size.

3.2.1 The Piecewise-Linear CDF

In the univariate case, the interpolation problem is easily solved by using a piecewise-linear approximation to the empirical CDF. The $n - 1$ gaps between the data values result in $n - 1$ piecewise-linear segments for the estimated CDF. If extrapolation in one or both tails is an issue, then the modeler can use Marsaglia's tail algorithm (Bratley, Fox, and Schrage, 1987) or kernel density estimation (Silverman, 1986).

In the bivariate case, the delineation of the support is less clear than in the univariate case. Using the rectangular support

$$\min\{x_1, x_2, \ldots, x_n\} \le x \le \max\{x_1, x_2, \ldots, x_n\},$$
$$\min\{y_1, y_2, \ldots, y_n\} \le y \le \max\{y_1, y_2, \ldots, y_n\}$$

for example, is likely to include regions of support that a modeler would want to exclude. In the method developed here, we use the convex hull of the data values as a preliminary support. (This support can be modified using techniques described subsequently.) We define the convex hull traditionally as the minimum convex set containing the data set of interest in the two-dimensional plane. The variate generation algorithm (Law, 2007, page 467) relies on conditioning:

1. generate $U_1 \sim U(0, 1)$
2. $X_0 \leftarrow F_X^{-1}(U_1)$
3. generate $U_2 \sim U(0, 1)$
4. $Y_0 \leftarrow F_{Y|X_0=x}^{-1}(U_2)$
5. return (X_0, Y_0)

We justify the algorithm with the following derivation. Consider the case where the joint CDF, $F_{X,Y}(x, y)$, is known and the joint density $f_{X,Y}(x, y)$ exists. The goal is to simulate the two-dimensional random vector (X, Y). Define the following variables.

1. Let U, V be independent $U(0, 1)$ random variables.
2. Let $X' = F_X^{-1}(U)$.
3. Let $Y' = F_{Y|X=X_0}^{-1}(V)$, where $X_0 = F_X^{-1}(U) = X'$. (Primes are used to distinguish the simulated random variables from the original random variables X, Y.)

Now use the change-of-variable formula (Hogg et al., 2005) to compute the joint density of (X', Y'). Recall the change-of-variable formula: if $(U, V) = S(X', Y')$, then

$$f_{X',Y'} = (f_{U,V} \circ S) \cdot |\det(DS)|.$$

In this case, $f_{U,V}$ is the constant function 1 on the unit square, so the term in parentheses is just 1. Solving for U and V in terms of X' and Y' above, we find that S is given by:

$$S : U = F_X(x'), V = F_{Y|X=F_X^{-1}(U)}(y') = F_{Y|X=x'}(y').$$

Then DS is the 2×2 matrix

$$\begin{bmatrix} f_X(x') & 0 \\ \star & f_{X,Y}(x', y')/f_X(x') \end{bmatrix},$$

where $\star$ denotes some irrelevant entry for the determinant. The lower-right entry follows by differentiating the formula

$$F_{Y|X=x'}(y') = \frac{\int_{-\infty}^{y'} f_{X,Y}(x', t)dt}{f_X(x')}$$

with respect to y'. Computing $|\det(DS)|$ gives $f_{X,Y}(x', y')$, so that (X', Y') does indeed have the same joint density as (X, Y).

The challenge associated with the development here is to find a reasonable nonparametric approximation to $F_{Y|X=x}^{-1}(\cdot)$. To illustrate the justification in using $F_{Y|X=x}^{-1}$, consider the scatterplot shown in Figure 3.1 with $x = 8$. The data indicate a wide range of potential values to generate for the second element of the random pair, y. Depending on the unknown bivariate population distribution this might be acceptable. However, given the observed data, it appears the associated y value should not potentially occupy this entire range, and might more appropriately be represented by the limits naturally occurring at the lower and upper intersections with the convex hull.

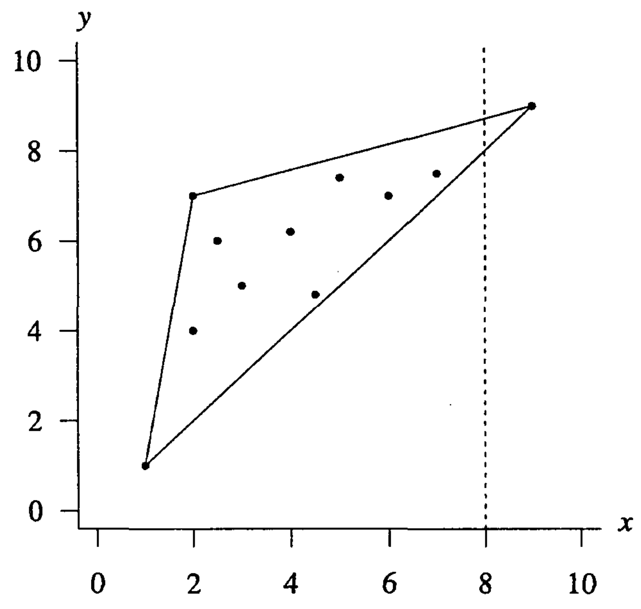


Figure 3.1: Intersection of a randomly generated $x = 8$ and the convex hull.

3.2.2 A Nonparametric Bivariate Generation Algorithm

By combining strategies used in the univariate case, an algorithm is devised to generate bivariate random variates from observed data pairs using a nonparametric heuristic approach. This algorithm requires a random sample of bivariate data drawn from an unknown continuous population distribution. A good algorithm produces variate pairs that adequately mimic the distribution associated with the observed data. If appropriate, the marginal data are moment matched at the beginning of the algorithm. The moment-adjusted vectors are created by first stretching the marginal data so that the variance of the piecewise linear CDF estimator matches that of the sample data variance, and then shifting the resulting marginal data values to match the marginal means. This process is only suitable in cases where the adjusted marginal values do not result in unrealistic data points, e.g., when service times are close to zero and adjusting them could produce impossible negative service times. The advantage of adjusting the data (when possible) is that the first two moments are conserved by the estimator, whereas, when the data is not adjusted, it is well known that the piecewise-linear CDF estimator's variance is less than the sample data variance. Matching the variances is important in computing the denominator in the correlation expression

$$Corr(X,Y) = \frac{Cov(X,Y)}{\sqrt{V(X)V(Y)}}.$$

Using the expressions derived Chapter 2, reprinted here, the ordered moment adjusted vector values $x'_{(j)}$ are calculated as

$$x'_{(j)} = x_{(1)} - \delta + w \sum_{j=1}^{i-1} g'_j,$$

where δ is the appropriate stretching parameter, w is the width of the support of the adjusted piecewise-linear CDF, and g'_j is a normalized gap value between sorted elements of the x vector. This calculation accomplishes matching the variance of the piecewise-linear CDF estimator to the sample data variance. We then match means by shifting each data value by

$$x''_{(i)} = x'_{(i)} - \left[\frac{x'_{(i)} + 2x'_{(2)} + \cdots + 2x'_{(n-1)} + x'_{(n)}}{2(n-1)} - \bar{x} \right].$$

The S-plus/R code for this moment matching process (designated as the `mm(x)` procedure) is provided in Appendix B. A more detailed explanation on matching the estimator's moments to the data is given in Chapter 2.

The algorithm is separated into a setup portion, and a generation portion. The terms x_i and y_i represent the observed data pairs, x'_i and y'_i are the moment adjusted data pairs, and lastly, (x'', y'') is the generated variate pair produced by the algorithm. The corresponding vectors are set in boldface.

Setup

1. $\boldsymbol{x}' \leftarrow \mathtt{mm}(\boldsymbol{x})$, $\boldsymbol{y}' \leftarrow \mathtt{mm}(\boldsymbol{y})$
2. $hull \leftarrow$ convex hull($\boldsymbol{x}'$, $\boldsymbol{y}'$)

Generation

1. generate $U \sim U(0,1)$
2. $x'' \leftarrow F_X^{-1}(U)$
3. $y_{\text{lo}} \leftarrow \text{minimum}\{hull(x'')\}$ (the height of lower intersection of the line $x = x''$ and the convex hull)
4. $y_{\text{hi}} \leftarrow \text{maximum}\{hull(x'')\}$ (the height of upper intersection of the line $x = x''$ and the convex hull)
5. $A \leftarrow \{i | y_{\text{lo}} < y_i' < y_{\text{hi}}\}$, $i = 1, 2, \ldots, n$ (the index set of interior points)
6. $w_k \leftarrow \dfrac{1}{1 + ((x_k - x'')/s)^2}$ for $k \in A$ where s is the sample standard deviation of $\boldsymbol{x}_A$, the set of interior points
7. $F_{Y|X=x} \leftarrow$ weighted piecewise-linear CDF conditioned on $x = x''$ (see Chapter 2 for details on creating the weighted piecewise-linear CDF)
8. generate $U \sim U(0,1)$
9. $y'' \leftarrow F_{Y|X=x}^{-1}(U)$

In step 6 of the generation portion of the algorithm, we include s to normalize the weight calculation. Data pairs with x_i values closer to the line $x = x''$ receive higher weight. Dividing the absolute difference $x_i - x''$ by s scales the factors in terms of standard deviation units.

This algorithm is nonconventional in the sense that it translates the data pairs directly into a variate generation algorithm, bypassing the usual density estimation step. There is, of course, an underlying joint probability density function associated with the algorithm. This joint probability density function is too tedious to calculate in general.

3.2.3 Applied Examples

Example 1. Consider the bivariate data set of size $n = 14$ random observations drawn from a continuous population: (4.1, 1.5), (6.2, 3.4), (8.3, 5.1), (7.8, 6.4), (5.2, 7.8), (2.0, 4.5), (1.9, 1.3), (2.7, 2.1), (3.5, 3.9), (4.0, 4.3), (3.6, 2.2), (4.4, 5.2), (5.0, 3.1), (5.3, 5.3).

Setup

1. Compute moment-matched $\boldsymbol{x}$ and $\boldsymbol{y}$ vectors, denoted as $\boldsymbol{x}'$ and $\boldsymbol{y}'$ for the data. Using the S-Plus/R `mm(x)` function, the adjusted vectors, to two decimal places, are: $\boldsymbol{x}' =$ (4.08, 6.48, 8.89, 8.32, 5.34, 1.67, 1.56, 2.47, 3.39, 3.96, 3.50, 4.42, 5.11, 5.45) and $\boldsymbol{y}' =$ (1.15, 3.35, 5.32, 6.82, 8.44, 4.63, 0.92, 1.85, 3.93, 4.39, 1.96, 5.44, 3.01, 5.55).
2. Find the convex hull of $\boldsymbol{x}'$ and $\boldsymbol{y}'$.

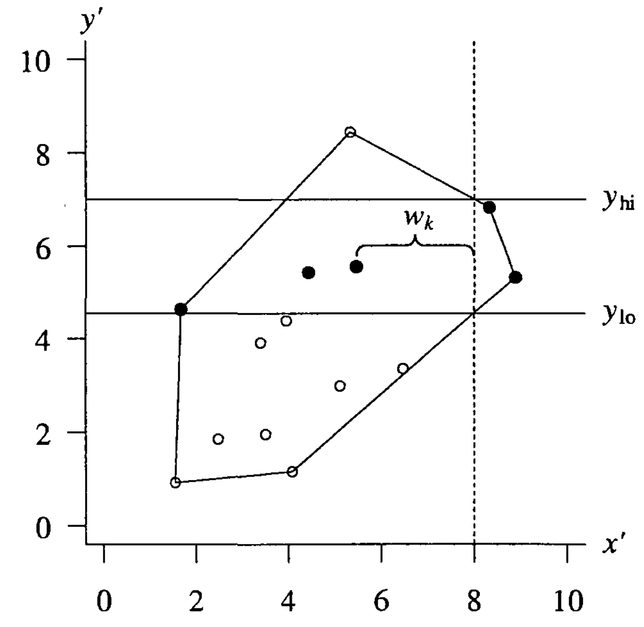


Figure 3.2: Plot of $\boldsymbol{x}'$ vs. $\boldsymbol{y}'$ and the convex hull.

Generation The S-Plus/R code provided in Appendix C combines the univariate strategies for generating from bivariate data. Figure 3.2 presents the adjusted bivariate data and associated convex hull. Using the piecewise-linear CDF created from the moment matched $\boldsymbol{x}'$ vector, the variate x'' is generated at $x'' = 8$. The vertical dashed line at $x'' = 8$ intersects the convex hull in exactly two places, denoted as y_{lo} and y_{hi} in the algorithm. The horizontal lines at these intersections establish the lower and upper limits capturing the interior original y data values used to create the weighted conditional piecewise-linear CDF for Y. The $|A| = 5$ interior values are the solid circles in Figure 3.2. These corresponding y values are appropriately weighted by w_k based on their respective horizontal distance from the vertical dashed line associated with x''. Using the weighted marginal piecewise-linear CDF created by the weighted interior y value, y'' is generated. Using this methodology, Figure 3.3 displays 50 variates from the original $n = 14$ data values where both the mean and variance for the piecewise-linear CDF's of $\boldsymbol{x}$ and $\boldsymbol{y}$ match that of the data.

The previous example illustrates the workings of the algorithm and associated results. Figure 3.3 shows (and the algorithm requires) that generated variates must lie within the convex hull created by the original data (if the data is adjusted to match moments, we can generate slightly outside the original convex hull since matching moments requires stretching each endpoint by a positive distance δ, and the interior points by a corresponding proportional distance). Additionally, if the user desires bivariate data for a certain region not encompassed by the observed data, it is only necessary to adjust the convex hull as desired. This feature allows significant advantages for studying specific aspects of a data set. For example, the user could easily develop cases for data analysis that include regions of interest while also including observed data. The next example illustrates the algorithm's ability to replicate multi-modal data in terms of means, variances, and correlation. Hörmann and Leydold (2000) highlight KDE's inability to accurately estimate multi-modal data, which makes meaningful

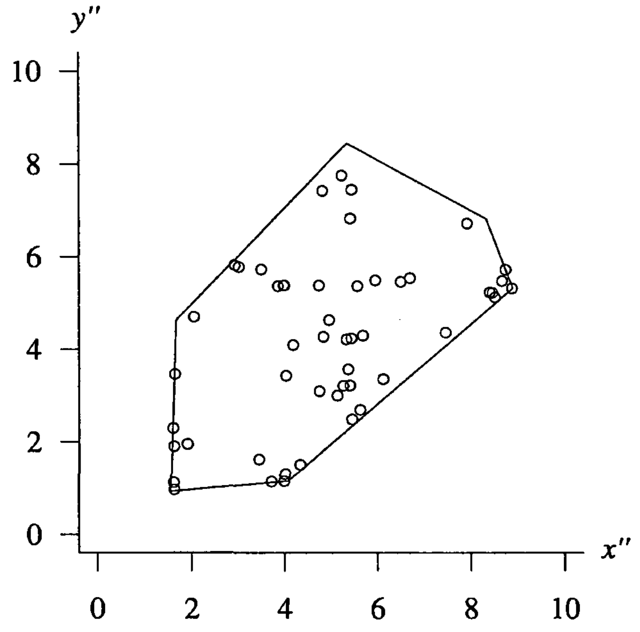


Figure 3.3: Plot of the marginal adjusted data, convex hull, and 50 random variates from $\boldsymbol{x}$ and $\boldsymbol{y}$.

variate generation impossible for such distributions.

Example 2. The Old Faithful geyser in Yellowstone Park is a commonly analyzed phenomenon. The data set (Weisberg, 1980) consists of $n = 299$ data pairs, the waiting time between eruptions (x_i) and the eruption duration (y_i), and is displayed in Figure 3.4, along with the convex hull. Though not easily visually distinguishable from the scatterplot, the data is tri-modal. Using a standard bivariate distribution to model this data set, such as the bivariate normal distribution, would not provide an adequate fit. For this data, it is appropriate to match the first two moments as doing so does not significantly change waiting nor duration times due to the large sample size. Additionally, matching the moments does not create any negative times.

Figure 3.5 shows the adjusted data and associated convex hull side-by-side

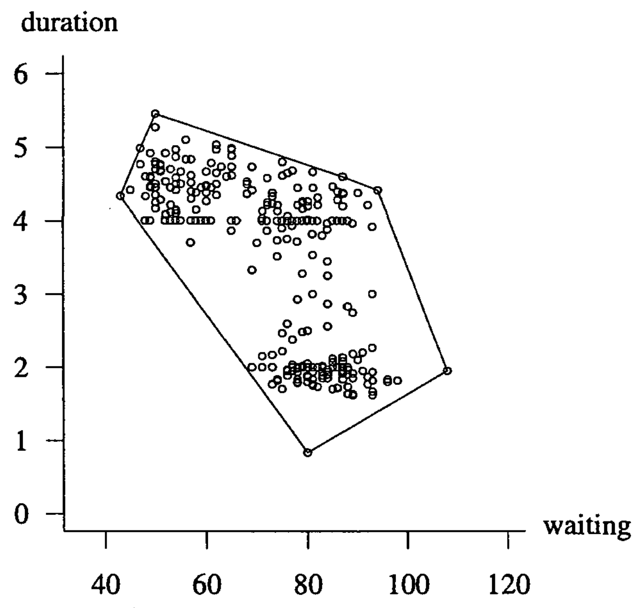


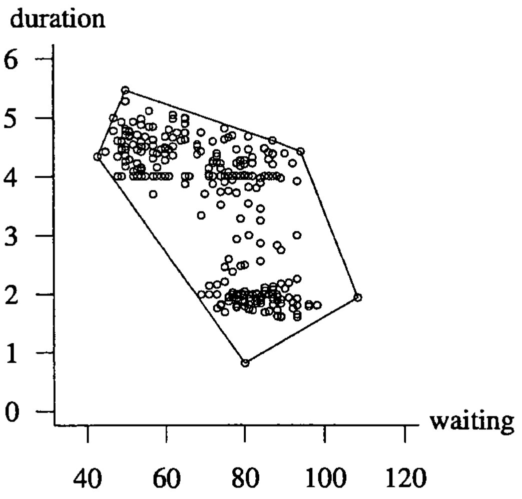
Figure 3.4: Plot of $n = 299$ waiting vs. duration times (minutes) for the Old Faithful Geyser.

with the variates generated by our algorithm. The first numerical column of Table 3.1 provides the sample statistics associated with the data, and the second column shows that the first and second moments, and the covariance are adequately conserved in the generated variates. The third column provides p-values for the hypothesis tests with t-tests used for the means and F-tests for the variances.

It is apparent that the algorithm will occasionally generate variates in

	$n = 299$ observed data	$n = 299$ generated data	p-value
avg waiting	72.31	73.24	0.407
avg duration	3.46	3.39	0.465
var waiting	192.94	183.18	0.654
var duration	1.32	1.42	0.529
covariance	−10.28	−9.07	

Table 3.1: Sample statistics for observed and generated data.



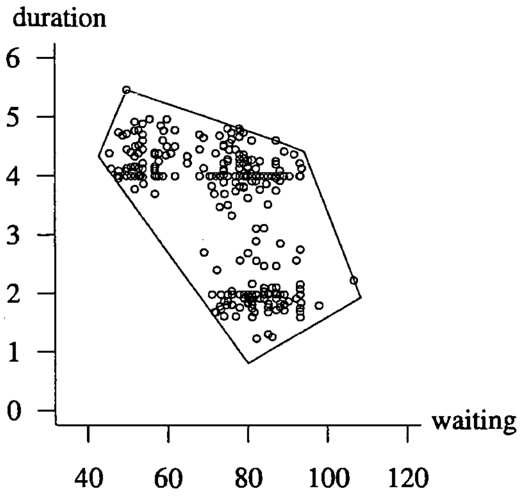


Figure 3.5: Sample adjusted observed data (left) and generated random variates (right) for waiting vs. duration times (minutes) for $n = 299$.

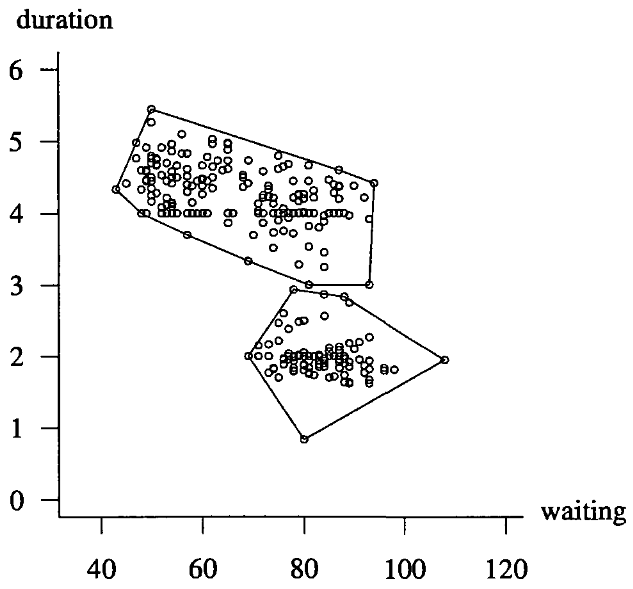


Figure 3.6: Plot of $n_1 = 105$ and $n_2 = 194$ waiting vs. duration times (minutes).

"white-space" (areas of the convex hull not represented by observed sample data values) of the convex hull as is expected. If this is problematic, we could fine-tune the appearance of the hull to avoid the possibility of these variates without significantly altering the algorithm. Alternatively, we could create two convex hulls as shown in Figure 3.6, with $n_1 = 105$ data values in the lower hull and $n_2 = 194$ data values in the upper hull. The algorithm is modified so that a bivariate pair is generated from the lower hull with probability 105/299 and the upper hull with probability 194/299. The algorithm's run time change for this adjustment is negligible.

3.3 Kernel Density Estimate Comparison

3.3.1 Generating Variates Via Kernel Density Estimation

Perhaps the most widely accepted method of univariate density estimation is kernel density estimation (KDE). The kernel density approximation of the underlying true distribution is defined as

$$\hat{f}_X(x) = \frac{1}{nb}\sum_{i=1}^{n} K\left(\frac{x - x_i}{b}\right)$$

where K is the kernel function, n is the sample size, and b is the bandwidth (smoothing) parameter. While several kernel functions exist in the literature the most commonly used kernel function is the Gaussian kernel,

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2} \qquad\qquad \infty < x < \infty$$

with mean zero and unit variance. These estimators provide a smooth density estimate with proven theoretical properties, making their choice of estimation a sound one. This estimator does not come without drawbacks. Using this estimator requires calculation of the bandwidth parameter b. Though many accepted versions of calculating b exist in the literature, choosing a method is not necessarily easy. For large values of b, oversmoothing occurs and important details about the underlying density may be lost. Additionally, if the parameter is too small, the resulting estimate is said to be undersmoothed and subtleties showing up in the density estimate caused by the sample data may lead to incorrect conclusions on the underlying distribution. We do not discount the importance of selecting an optimal bandwidth parameter, but will focus on the KDE method for comparison to the proposed algorithm. We reference Hörmann and Leydold (2000) for use of kernels (and selection of a smoothing parameter) in generating bivariate data from an observed sample. They provide an efficient algorithm for sampling from a multi-dimensional kernel density estimate. Using their algorithm with a normal kernel function, generating variates is very fast. The algorithm is divided into a setup and generation portion.

Setup

For a random sample $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n$ of d length vectors, compute:

1. the mean vector $\overline{\boldsymbol{x}}$,
2. the estimated covariance matrix Σ,
3. the Cholesky factor $\boldsymbol{l}$ of Σ,
4. the smoothing parameter b,
5. the variance correction factor c_b.

Generation

1. Generate a random integer I uniformly distributed on $\{1, 2, \ldots, n\}$.
2. Generate a random vector $\boldsymbol{W}$ of d independent normal variates.
3. Return $\boldsymbol{Y} = \overline{\boldsymbol{x}} + (\boldsymbol{X}_I - \overline{\boldsymbol{x}} + \boldsymbol{l}\,(b\boldsymbol{W}))\,c_b$.

In this algorithm a full covariance matrix is specified from the observed data. Using the Old Faithful geyser data (Weisberg, 1980), the estimated covariance matrix, Σ, is

$$\Sigma = \begin{bmatrix} 192.94 & -10.28 \\ -10.28 & 1.32 \end{bmatrix}.$$

The joint density is estimated as a sum of $n = 299$ translated versions of the chosen kernel function (bivariate normal in this case) multiplied by $\frac{1}{nb}$. Though there are many accepted versions of calculating b for the univariate case, the multidimensional case is more challenging. Silverman (1986) suggests a simple calculation for b as

$$b = \left(\frac{4}{(d+2)n}\right)^{1/(d+4)},$$

where d is the dimension of the data. Thus the bivariate case results in

$$b = \left(\frac{1}{n}\right)^{1/6}.$$

Additionally, a variance correction factor is included because the variance of the empirical distribution is always larger than the variance of the observed data (Silverman, 1986). Hörmann and Leydold (2000) define the variance correction as c_b, where

$$c_b = \sqrt{1+b^2}.$$

3.3.2 Comparisons for Unknown Joint Densities

To compare the two variate generation methods, 100 replications were made, each of size $n = 299$ variates, using the geyser data introduced earlier. Prior to the study it was determined that a single replication is considered acceptable if it successfully captures the tri-modal KDE density appearing in Figure 3.7.

This density was computed directly from the $n = 299$ data pairs using S-Plus/R as described in Bowman and Azzalini (1997) for a normal kernel function and a normal optimal smoothing parameter. These estimated joint density plots are only used as a visual tool for comparing variate generation methods. The methods compared are (1) nonparametric algorithm for unadjusted waiting and duration times, (2) nonparametric algorithm for adjusted waiting and duration times, and (3) Hörmann and

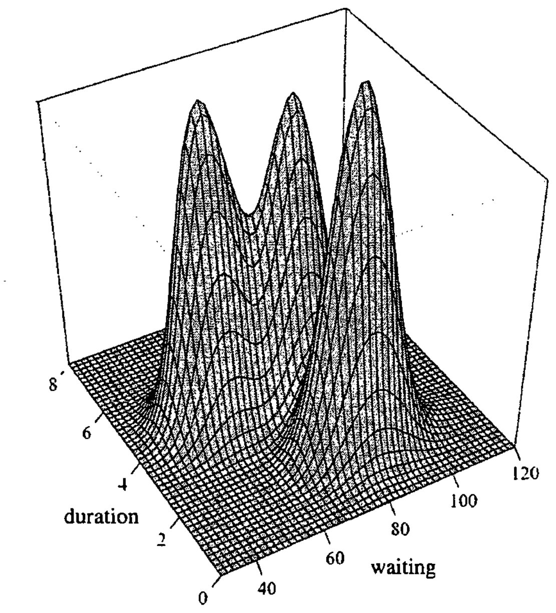


Figure 3.7: Joint density estimate of Old Faithful geyser data.

Leydold's variance-corrected KDE algorithm. For each method and replication a three-dimensional estimated joint density plot like the one shown in Figure 3.7 was inspected for a tri-modal density. Methods one and two (those proposed in this chapter) always captured the tri-modal appearance, while the KDE algorithm failed 35 times out of 100. An example of a failure instance is depicted in Figure 3.8.

Recognizing that the chosen smoothing parameter in the KDE algorithm is "oversmoothing" due to the multi-modality of the distribution, the parameter value was reduced by half as suggested in Hörmann and Leydold (2000) and the experiment was repeated. Doing so resulted in six failures out of 100 replications. This reduction in failures is evidence of the estimated density's sensitivity to the smoothing parameter selection. And, while the generated density estimate improved dramatically, it is still outperformed by the proposed completely nonparametric algorithm.

The next example consists of warranty claim data provided by General Motors for model year 2000 cars sold in the month of December, 2000. The bivariate data

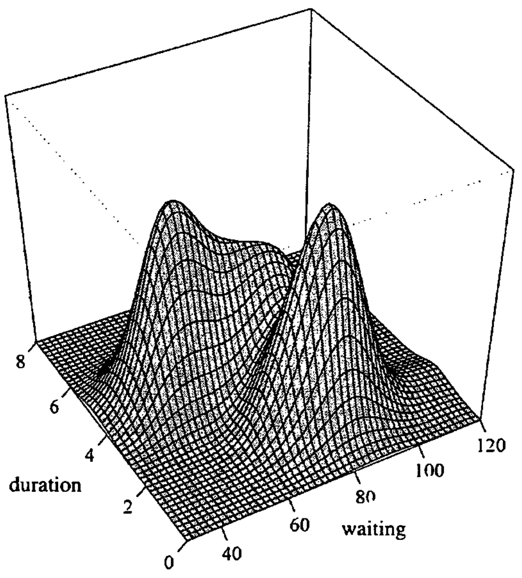


Figure 3.8: Example of a failure instance for the KDE joint density estimate.

values are the mileage and the age of the vehicle at warranty claim. All vehicles share a three-year (1095 day), 36,000 mile warranty. This data set is unique because it is bounded below at zero and above at three years/36,000 miles. Given the lower and upper bounds on the data, it is inappropriate to stretch the data and match moments as on the geyser data. A scatterplot of the data is provided in Figure 3.9, and the corresponding three-dimensional density estimate in Figure 3.10. The figures depict a pronounced mode close to the origin and a less prevalent mode near the mileage axis upper bound. This is logical because a buyer might not recall when a three-year warranty will expire, but can easily notice the approaching 36,000 mile warranty limit. General Motors might be interested in the impact of adjusting warranty durations.

Using the same type study as the geyser data, we test the proposed variate generation algorithm against both the variance-corrected KDE and reduced smoothing parameter variance-corrected KDE sampling techniques. Figure 3.11 depicts one resulting joint density comparison instance. Once again, it is apparent that variance-

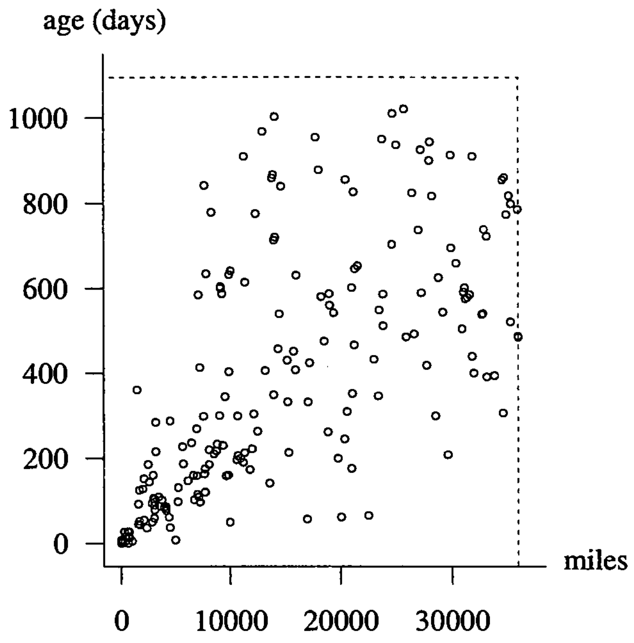


Figure 3.9: Scatterplot of miles vs. age (days) at warranty claim, $n = 259$.

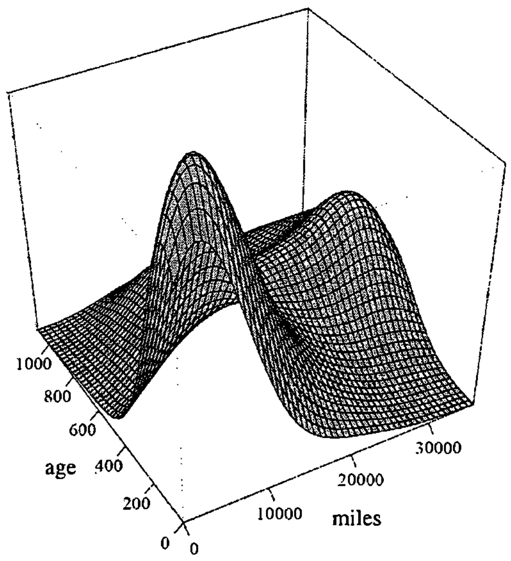


Figure 3.10: KDE joint density estimate of miles vs. age.

corrected KDE "oversmooths," while the reduced smoothing parameter KDE performs better in estimating the observed warranty data as depicted in Figure 3.12.

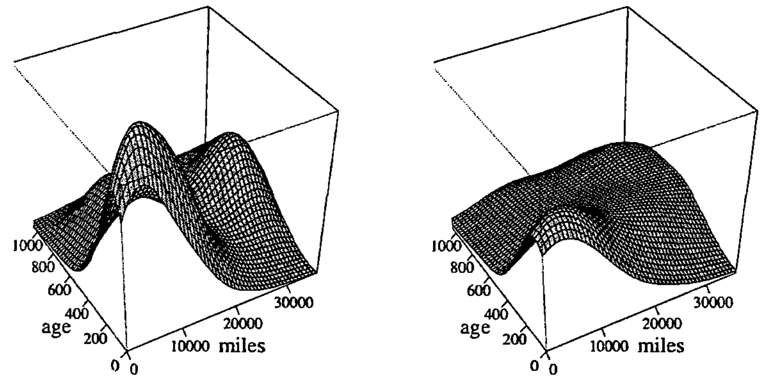


Figure 3.11: Variate generation for the proposed algorithm vs. variance-corrected KDE.

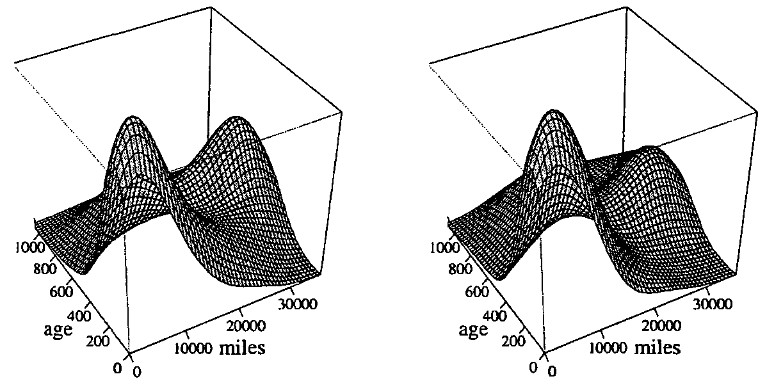


Figure 3.12: Variate generation for the proposed algorithm vs. reduced smoothing KDE.

A scatterplot of the KDE variance-corrected results, shown in Figure 3.13, displays the tendency of KDE to generate more densely at the pronounced mode, further accentuating the possibility of variates outside of the support rectangle when the mode is close to zero, as is the case in this example. In addition, variates are also produced that lie outside the upper bounds for mileage and age. This behavior is troublesome for KDE and not easily overcome without resorting to some type of acceptance–rejection or thinning method. Both of these options ruin synchronization, which might be needed if a variance-reduction technique is employed.

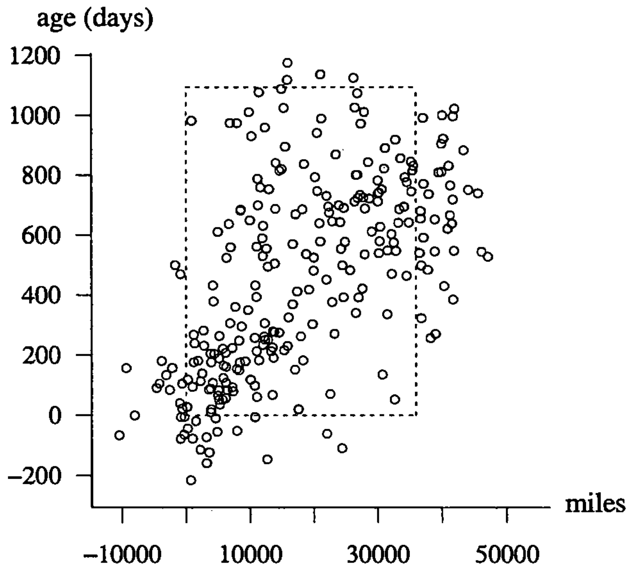


Figure 3.13: Scatterplot of variates generated via KDE.

The range of variates produced by the two approaches further accentuates their differences. Table 3.2 lists the minimums and maximums for each approach, along with the percentage of realizations falling outside the allowable warranty bounds. Given that all the generated variates for the proposed algorithm must (by construction) fall within the allotted bounds, impossible variates cannot occur. Consequently, using the KDE sampling method requires discarding impossible variates. Finally, visual comparisons of 100 joint densities for each approach resulted in the proposed algorithm dominating KDE in capturing the original data's depiction of customer warranty claims.

Using the normal kernel poses difficulty in modeling bounded data in two dimensions, as well as capturing multi-modal behavior. In months where sales numbers are higher, the upper limits of mileage and age are even more densely covered, further

	min miles	max miles	min age(days)	max age(days)	percent < 0	percent $> 3/36K$
observed data	8	35993	0	1056	0.0	0.0
proposed algorithm	14	35983	0	1047	0.0	0.0
var. corr. KDE	−11093	53178	−156	1104	18.5	7.0
reduced sm. param. var. corr. KDE	−2907	39984	−34	1179	8.5	6.2

Table 3.2: Range and percentage of variates outside allowable bounds.

exhibiting multi-modal behavior.

In the proposed variate generation algorithm, the modeler has the choice between using the convex hull associated with the data pairs or using the rectangle with opposite corners $(0, 0)$ and $(36000, 1095)$. Figure 3.9 shows that there will be a significant difference between these two choices.

3.3.3 Comparisons for Known Joint Densities

We will now compare KDE and the proposed algorithm for two known joint densities, the first of which has infinite support and the second with bounded support.

The first example is an equiprobable mixture of three bivariate normal distributions, with parameters as indicated in Table 3.3.

	Bivariate normal parameters		
μ_X	2	4	8
μ_Y	1	8	4
σ_X	1	1	2
σ_Y	2	1	2
ρ	1/5	−1/5	−1/3

Table 3.3: Parameters for three equiprobable bivariate normals.

Using this mixture as the underlying density, $n = 150$ variates were generated for use as the observed sample data. We then compare standard KDE with a Gaussian kernel (bandwidth parameter is $1/n^{1/6}$) and the proposed algorithm for 150 generated variates. Figure 3.14 illustrates the observed data in the left-hand plot, the KDE generated estimate on the right and the proposed algorithm's estimate in the center. A visual inspection indicates oversmoothing in the KDE case, a situation that could be remedied through manipulation of the smoothing parameter. Further work with the smoothing parameter did refine the KDE estimate suitably, and as expected, given a mixture of bivariate normals, KDE does well with proper selection of the smoothing parameter.

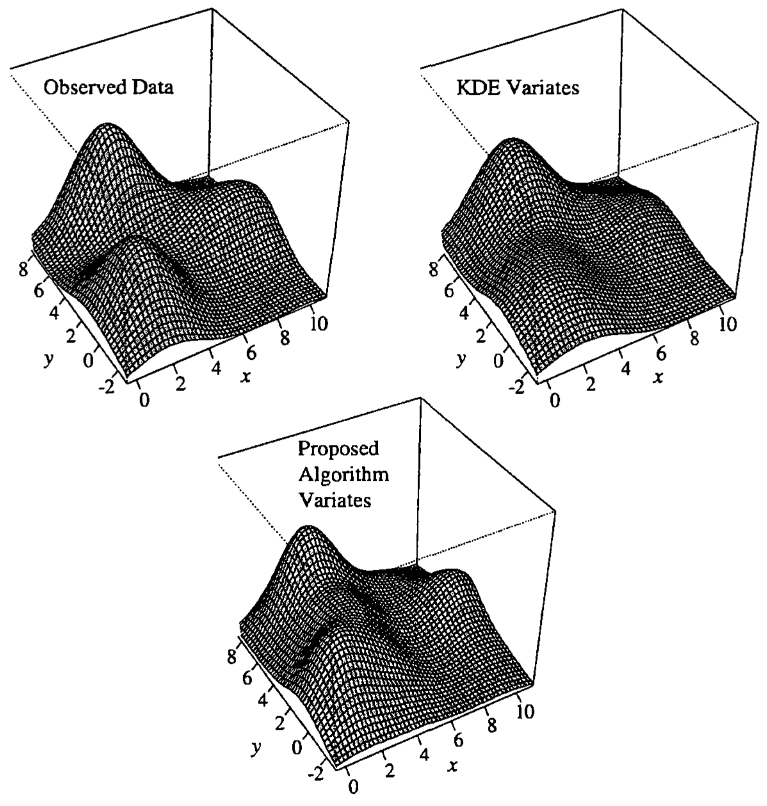


Figure 3.14: Observed data (left) and density estimate comparisons for KDE (right) and the proposed algorithm (center).

As a second example, consider a uniform bivariate distribution with uniform support on the unit square. We will use this example to illustrate how our algorithm performs in the limit with regard to the marginals, which in this case are bounded by

$(0, 1)$. The experiment consists first of generating $k = 20, 50, 100$ data pairs from the bivariate uniform. Using these k data pairs, we then exercise the proposed algorithm and KDE, generating a single $m = 1$ two-dimensional variate for each. We repeat this experiment 100,000 times and check the resulting marginal densities which we would like to converge to the theoretical marginals, each $U(0, 1)$. Figure 3.15 shows the resulting marginal densities for the proposed algorithm using $k = 50$ observed data pairs. The left-hand plot indicates that the density appears to converge to $U(0, 1)$ as desired. However, the conditioned density clearly does not. This result occurs because of the algorithm's tendency to designate more mass where the generated x value intersects the convex hull of the observed data. So, even though we replicate the experiment many times, there is the tendency to not adequately cover the vertical axis toward the upper and lower limits.

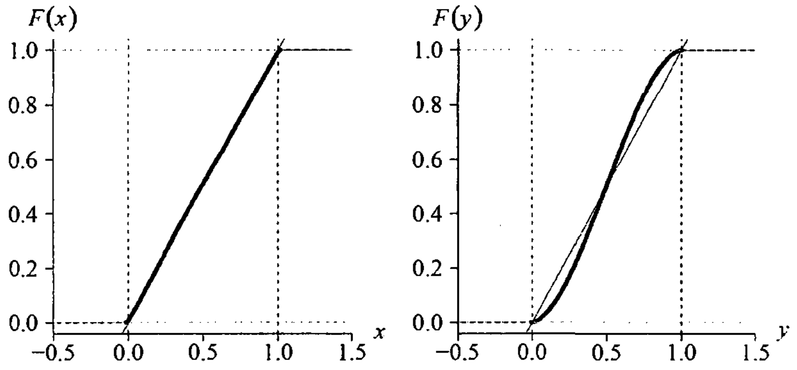


Figure 3.15: Estimated marginal densities for the unit square bivariate uniform distribution using the proposed algorithm.

A suitable manipulation of the algorithm allows us to partially correct this shortcoming by spreading the error equally between x and y. Since the vertical axis suffers in marginal estimation, we can modify the algorithm by alternating the roles of x and y on each subsequent (x, y) pair generated. Figure 3.16 depicts the estimated marginal densities after manipulating the algorithm.

In the General Motors example, the support is rectangular, and furthermore,

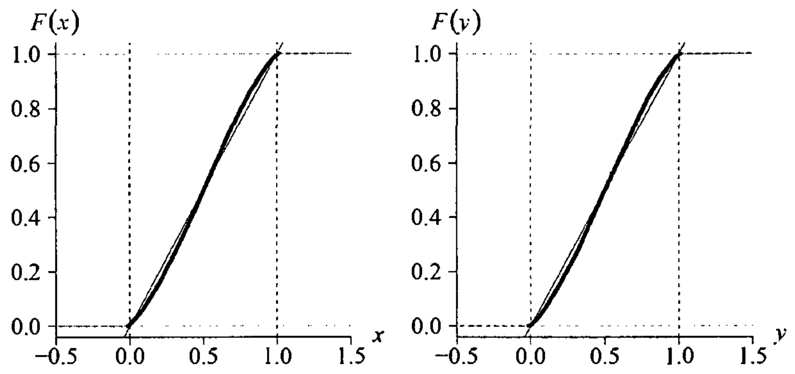


Figure 3.16: Estimated marginal densities for the unit square bivariate uniform distribution using the alternating algorithm.

known. In this case we could have artificially created the convex hull limits since the minimum and maximum for each marginal is known and fixed. For this example, we now fix the support as the unit rectangle, thus the convex hull is $(0,1) \times (0,1)$. We will run two cases for the fixed support, the proposed algorithm and the alternating algorithm. Given that the hull is fixed for both cases, the corresponding results do not differ significantly. Figure 3.17 shows the marginals for the first case, the proposed algorithm.

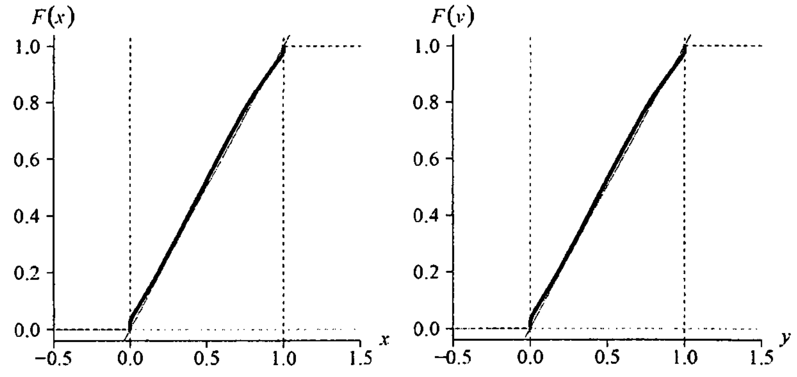


Figure 3.17: Estimated marginal densities for the unit square bivariate uniform distribution with fixed support $(0,1) \times (0,1)$.

Lastly, we perform the same experiment for KDE, again using a Gaussian kernel and the same smoothing parameter used earlier. Figure 3.18 shows that although KDE does well over most of the support, it also suffers at the lower and upper end of the support. Furthermore, the KDE method of course generates a substantial number of impossible variates.

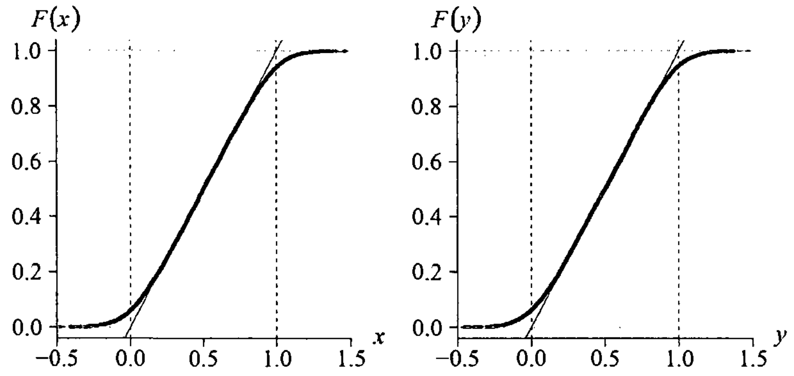


Figure 3.18: Estimated marginal densities for the unit square bivariate uniform distribution using KDE.

Table 3.4 displays the squared error between the CDF and $N = 100,000$ generated data points, calculated as

$$\frac{1}{N}\sum_{i=0}^{N}\left(\hat{F}(x_i) - F(x_i)\right)^2,$$

where $\hat{F}(x_i)$ is the estimated marginal CDF value at x_i and $F(x_i)$ is the theoretical CDF value at x_i. As another measure, we could include a quantile comparison, however, other than the lower and upper quantile discrepancies for KDE, there does not seem to be much difference across the board. As expected, KDE performs well throughout, except for the impossible variates generated. We could also change the kernel to a distribution with fixed support, which would reduce the extremity to which KDE produces impossible variates. However, the inclusion of such a comparison does not substantially change the overall results.

Since it is impossible to generate variates exactly from some data set without knowing the underlying distribution, questioning the quality of the variates generated from some known parametric distribution is justified. These hypothetical examples show that using the proposed algorithm exhibits quality at least as good as KDE. In terms of generation speed, KDE has the advantage over the proposed algorithm. In testing the vectorized version of the proposed algorithm's code versus KDE, excluding setup, we find that KDE runs about twice as fast, and given that the proposed algorithm's run time is a function of n, KDE's advantage is more pronounced for large sample sizes.

$n = 20$	X error	Y error	correlation
Proposed Algorithm	3.7×10^{-5}	3.3×10^{-3}	0.007
Alternating Algorithm	6.8×10^{-4}	6.7×10^{-4}	0.008
Fixed Support Algorithm	1.3×10^{-3}	2.2×10^{-3}	0.022
Alternating Fixed Support Algorithm	3.9×10^{-4}	4.1×10^{-4}	0.012
KDE Algorithm	5.7×10^{-4}	5.3×10^{-4}	-0.002
$n = 50$	X error	Y error	correlation
Proposed Algorithm	1.1×10^{-5}	3.8×10^{-3}	0.001
Alternating Algorithm	9.0×10^{-4}	8.5×10^{-4}	-0.001
Fixed Support Algorithm	2.8×10^{-4}	2.2×10^{-3}	0.015
Alternating Fixed Support Algorithm	3.3×10^{-4}	3.0×10^{-4}	0.009
KDE Algorithm	3.7×10^{-4}	3.6×10^{-4}	0.004
$n = 100$	X error	Y error	correlation
Proposed Algorithm	3.2×10^{-6}	3.8×10^{-3}	$-.001$
Alternating Algorithm	8.9×10^{-4}	9.1×10^{-4}	-9.2×10^{-5}
Fixed Support Algorithm	7.7×10^{-5}	2.6×10^{-3}	-6.6×10^{-4}
Alternating Fixed Support Algorithm	4.7×10^{-4}	4.8×10^{-4}	-7.1×10^{-4}
KDE Algorithm	2.6×10^{-4}	2.5×10^{-4}	-0.006

Table 3.4: Marginal CDF squared error for the estimates of the bivariate uniform distribution.

3.4 Limitations

There are a number of limitations associated with the proposed algorithm which we outline in this chapter. The three limitations discussed here are

1. The lack of an expression for the nonparametric joint PDF,
2. The algorithm's performance relative to KDE,
3. The speed of the algorithm.

Though the proposed nonparametric algorithm is "blackbox" in that no decisions are required by the modeler, there is an underlying joint PDF. The algorithm goes directly from data to random variates, bypassing the usual step of specifying the PDF because of its complicated nature for large values of n. For small n, however, the joint PDF is easily available, as illustrated in the example below.

Example 3. Consider a data set that consists of just three ($n = 3$) noncollinear pairs. The convex hull is the triangle with the data pairs as vertices and no data values are internal to the convex hull. More specifically, consider the data pairs

$$(1,1), (2,1), (3,3).$$

Exploiting the fact that the x-values are equally spaced, the variate generation algorithm reduces to

```
generate $X \sim U(1,3)$
if $(X < 2)$ generate $Y \sim U(1,X)$
else         generate $Y \sim U(2X-3,X)$
```

The joint PDF of X and Y associated with this algorithm can be determined as the product of the marginal distribution for X

$$f_X(x) = 1/2 \qquad 0 < x < 2$$

and the conditional distribution

$$f_{Y|X=x}(y|x) = \begin{cases} \dfrac{1}{x-1} & 1 < y < x;\ \ 1 < x < 2 \\ \dfrac{1}{3-x} & 2x-3 < y < x;\ \ 2 \le x < 3 \end{cases}$$

yielding

$$f_{X,Y}(x,y) = \begin{cases} \dfrac{1}{2(x-1)} & 1 < y < x;\ \ 1 < x < 2 \\ \dfrac{1}{2(3-x)} & 2x-3 < y < x;\ \ 2 \le x < 3. \end{cases}$$

Characteristics of this joint distribution are consistent with variates generated by the algorithm. The covariance of X and Y, for example, is 1/3, and the sample covariance of variates generated by the algorithm converges to 1/3 as the number of data pairs generated tends to infinity.

Second, the algorithm's performance is compared with that of KDE. There are few bivariate parametric distributions where variate generation is easy. We use the bivariate normal distribution to compare the impact of correlated random variables on the proposed algorithm and KDE. We expect KDE to perform extremely well since the underlying distribution is bivariate normal. The infinite tails associated with the bivariate normal distribution give an advantage to KDE, just as a bounded region, such as in the General Motors warranty data case, gives an advantage to the proposed algorithm. For the study, the underlying distribution is given by the parameters

$$\mu_X = 1, \mu_Y = 2, \sigma_X = 4, \sigma_Y = 3, \rho = 0.01, 0.99,$$

where ρ varies from extremely low to high correlation. In addition to these two extreme values of ρ, the same study considered intermediate values of ρ, however including them here is not informative. For each value of ρ, we chose $n = 100$ and

$n = 200$ as the observed sample sample sizes from the bivariate normal distribution. For each of the observed sample sizes, $N = 10$ and $N = 40$ variates were generated using KDE and the proposed algorithm (with and without moment matching). These variates were then used to calulate confidence intervals for the means, standard deviations, and correlation. The experiment for each (n, N) pair combination was conducted 10,000 times and the count of confidence intervals containing each of the five parameters were tallied. Using the F distribution associated with the Clopper–Pearson confidence interval considered not significantly different from 9,500 and $\alpha = 0.01$, the confidence interval counts are in the interval [9443, 9555]. Tables 3.5 and 3.6 contain the results of the simulation study, where boldface numbers are in [9443, 9555].

n	N	μ_X	μ_Y	σ_X	σ_Y	ρ	algorithm
100	10	9413	9304	**9496**	8722	**9513**	moment matched algorithm
		9384	9285	**9514**	8468	**9503**	proposed algorithm
		9392	9403	9112	9097	9396	KDE
100	40	9041	8652	9188	4921	9213	moment matched algorithm
		9038	8638	9089	3937	9204	proposed algorithm
		9113	9097	8068	8109	9079	KDE
200	10	9438	9418	**9528**	8879	**9554**	moment matched algorithm
		9421	9390	**9552**	8752	**9523**	proposed algorithm
		9445	**9447**	9205	9230	9400	KDE
200	40	9291	9111	9382	5331	**9451**	moment matched algorithm
		9251	9100	9391	4759	9390	proposed algorithm
		9298	9300	8650	8568	9313	KDE

Table 3.5: Confidence interval count for bivariate normal parameters and $\rho = 0.01$.

For low correlation, where we expected KDE to perform extremely well, the results do not indicate KDE dominating the proposed algorithm for producing variates that properly mimic the five distribution parameters. On the contrary, KDE performs rather poorly; in several instances it is outperformed by the proposed algorithm, regardless of whether moments are matched. These results were a bit disappointing for KDE, given the underlying distribution is bivariate normal with almost zero correlation. One weakness of the proposed algorithm is apparent in the σ_Y column. This weakness can be partially overcome by implementing the alternating algorithm described in the previous chapter.

n	N	μ_X	μ_Y	σ_X	σ_Y	ρ	algorithm
100	10	9407	9388	**9483**	**9463**	9399	moment matched algorithm
		9400	9394	**9493**	**9468**	9393	proposed algorithm
		9420	9401	8711	9386	861	KDE
100	40	9063	9051	9161	9121	7801	moment matched algorithm
		9010	9001	9088	9033	7682	proposed algorithm
		9123	9034	7132	9106	0	KDE
200	10	**9458**	**9443**	**9549**	**9531**	9344	moment matched algorithm
		9432	9431	**9539**	**9521**	9347	proposed algorithm
		9450	**9466**	8967	**9478**	1304	KDE
200	40	9291	9255	9382	9334	7094	moment matched algorithm
		9224	9212	9336	9294	7082	proposed algorithm
		9319	9285	7756	9261	0	KDE

Table 3.6: Confidence interval count for bivariate normal parameters and $\rho = 0.99$.

There is no surprise that KDE had trouble inducing extremely high correlation, however, we did expect KDE to perform better in capturing the other distribution parameters. The proposed algorithm clearly outperformed KDE in inducing correlation. Using another distribution that is bounded below (e.g., the bivariate exponential) would be even more troublesome for KDE because it would produce negative variates. Choosing to reject the negative variates (which would be required in variate generation) induces bias. The same problem exists in KDE for the car mileage and time data presented earlier in which both X and Y were bounded above and below.

The final limitation noted for the proposed algorithm is generation speed. While generating the first element of the random variate pair is fast, generating the second element requires creating the conditional piecewise-linear CDF, which is slow for large n. However, the algorithm benefits when high correlation exists in the observed variate pairs. High correlation results in a tight convex hull where once the first element of the random pair is generated, the conditional piecewise-linear CDF may involve only a small number of data points, or in the most extreme case, may be uniformly distributed between the points where the x variate intersects the convex hull.

3.5 Conclusions

A nonparametric method of generating bivariate data was presented with examples in this chapter. The method is blackbox, synchronized, and effectively captures multi-modal two-variable dependencies for most data sets. The method does not require any information about the underlying distribution of the empirical data, nor does it require joint density estimation as an intermediate step for variate generation. Thus, given an appropriate observed bivariate data set, a researcher is capable of generating variates without the risk of introducing error associated with generating from some incorrect parametric distribution. Given continuous bivariate data, this method is capable of producing variates efficiently, and, in the case of observed data falling into recognizable groups, the algorithm can be easily altered for suitable employment. In a comparison study, the method performs at least as well as an accepted KDE generation algorithm in terms of estimation quality for selected data sets. Three significant contributions of the proposed algorithm are (1) it is completely nonparametric and requires no parameters from the modeler, (2) it is simple to implement, and (3) it is a one-to-one (synchronized) variate generation algorithm whose resulting random vectors are capable of representing multi-modal bivariate distributions and will not produce impossible variates for fixed supports. In summary, the algorithm's advantages over sampling from a KDE algorithm are

- no reliance on selected kernel density function
- no reliance on selected smoothing parameter
- cannot produce unrealistic variates (e.g., negative times from a service time

distribution).

Three decisions are required from the modeler that are dependent on the data set. First, the modeler must decide if the data should be stretched in order to match moments. Second, the modeler must decide whether to use the convex hull associated with the (stretched or raw) data, or use a rational convex hull as in the case of the warranty data. Finally, the modeler must decide whether a single convex hull, as in Figure 3.4, or multiple convex hulls, as in Figure 3.6, is appropriate.

Chapter 4

Control Chart Constants for Non-Normal Sampling

4.1 Introduction

Control charts are widely used in industry to provide insight on process behavior and identify assignable causes associated with a shift in the mean value of the process. These charts were first proposed in a memo by Walter Shewhart in 1923 at Bell Telephone Laboratories. To create the control limits, estimates for the mean and standard error of the population are required, along with constants that serve as bias correction factors (Shewhart, 1980). The first control chart constants, then denoted by d_2 and d_3 (for the sample range), were proposed by Tippett (1925). McKay and Pearson (1933) obtained the exact distribution of the sample range for $n = 3$ observations drawn from a normal distribution. Hartley and Pearson (1951) tabulated the fractiles of the mean of the sample range for $n = 2$ to $n = 20$ (Wheeler, 2000). The terms *bias correction factor* and *control chart constant* are used interchangeably.

Bias correction factors for standard deviations followed a similar development. They too are based on an underlying normal distribution. For both sets of constants, extensive work exists (Wheeler, 2000) showing the robustness of these constants for data from non-normal distributions. For the most part, similar constants for non-normal distributions do not appear in the literature for two reasons: (1) most applications involve sampling from normal populations, and (2) they are not easily computed. The purpose of this chapter is to offer an alternative method of computation using A Probability Programming Language (APPL), developed by Glen et al. (2001), to compute the exact values of these control chart constants. Additionally, APPL typically provides exact results rather than approximations. Although normal sampling can be assumed in the vast majority of statistical process control applications, occasions will arise where non-normal sampling is an appropriate assumption. The development here allows an engineer to easily obtain the appropriate control chart constants in these alternate settings.

4.2 Constants d_2, d_3

The aforementioned constants d_2 and d_3 relate to the distribution of the sample range, denoted by R. The correction factor d_2 is a function of the mean of the sample range and the population standard deviation. Given a random sample $X_1, X_2, \ldots, X_n$ from a population with cumulative distribution function $F(x)$, probability density function $f(x)$, finite unknown variance σ_X^2, and associated order statistics $X_{(1)}, X_{(2)}, \ldots, X_{(n)}$, the sample range, R, is

$$R = X_{(n)} - X_{(1)}. \tag{4.1}$$

The joint probability density function of the order statistics $X_{(i)}$ and $X_{(j)}$ associated with a sample size n given by Hogg et al. (2005) is

$$\begin{aligned} f_{X_{(i)},X_{(j)}}(x_{(i)}, x_{(j)}) = \frac{n!}{(i-1)!(j-i-1)!(n-j)!} \left[F(x_{(i)})\right]^{i-1} \left[F(x_{(j)}) - F(x_{(i)})\right]^{j-i-1} \\ \times \left[1 - F(x_{(j)})\right]^{n-j} f(x_{(i)}) f(x_{(j)}) \qquad x_{(i)} < x_{(j)} \end{aligned}$$

for integers $1 \le i < j \le n$. For $i = 1$, $j = n$, this simplifies to

$$f_{X_{(1)},X_{(n)}}(x_{(1)}, x_{(n)}) = n(n-1)\left[F(x_{(n)}) - F(x_{(1)})\right]^{n-2} f(x_{(1)}) f(x_{(n)}) \qquad x_{(1)} < x_{(n)}. \tag{4.2}$$

Burr (1967) uses a change of variable, $X_{(n)} = X_{(1)} + R$ (since, by definition $R = X_{(n)} - X_{(1)}$) in (4.2) to find the joint density of $X_{(1)}$ and R and then integrates out $X_{(1)}$ to find the probability density function of R. This, of course, works well for distributions with closed form cumulative distribution functions; however, cumulative distribution functions involving mathematically intractable integrals are problematic. Once the distribution of R is obtained, it is used it to correct bias by

$$E[R] = d_2 \sigma_X. \tag{4.3}$$

Burr (1967) also suggests an easier approach to find $E[R]$, which lends itself well to implementation in APPL. Using (4.1), for a sample of size n, the expected value of the sample range is

$$E[R] = E\left[X_{(n)}\right] - E\left[X_{(1)}\right], \tag{4.4}$$

therefore, using (4.3) and (4.4), we can express d_2 as

$$d_2 = \frac{E[R]}{\sigma_X} = \frac{E\left[X_{(n)}\right] - E\left[X_{(1)}\right]}{\sigma_X}.$$

This result can be implemented using the APPL `RangeStat` procedure for select distributions. This procedure returns the distribution of the sample range for a sample of size n. Equivalently, we can use the `OrderStat` procedure, and return d_2 values exactly. For sampling from a normally distributed population we can always remove the mean by subtraction, resulting in a random variable with mean zero. For $n = 3$ consider the APPL statements

```
> n := 3:
```

```
> X := NormalRV(0, sigma):
> (Mean(OrderStat(X, n, n)) - Mean(OrderStat(X, n, 1)))
    / sqrt(Variance(X));
```

which yield the exact value of $d_2 = 3/\sqrt{\pi}$. Though this is convenient, APPL is only capable of returning the exact symbolic expression of d_2 for $n = 2$ and $n = 3$. For $n > 3$, the problem is mathematically intractable and the integrals must be evaluated numerically. However, if population distribution parameter values are input for the code above, APPL is capable of solving for d_2 when $n \geq 3$. Since d_2 depends only on n (and is independent of μ, σ), assigning values to these distribution parameters does not affect d_2.

We proceed in a similar manner for d_3, which corrects for the standard deviation of the range. The relationship is

$$\sigma_R = d_3 \sigma_X \qquad \Rightarrow \qquad d_3 = \frac{\sigma_R}{\sigma_X}.$$

Since APPL can compute the exact distribution of R, we can also obtain σ_R easily for select distributions.

Example 1. Given that X_1, X_2, and X_3 are iid exponential(λ) random variables, find the bias correction factors d_2 and d_3 for the sample range. The APPL statements

```
> n  := 3:
> X  := ExponentialRV(lambda):
> R  := RangeStat(X, n):
> d2 := Mean(R) / sqrt(Variance(X));
> d3 := sqrt(Variance(R)) / sqrt(Variance(X));
```

yield

$$d_2 = 3/2 \qquad \text{and} \qquad d_3 = \sqrt{5/2} \cong 1.118.$$

Likewise, when $n = 18$,

$$d_2 = \frac{42142223}{12252240} \cong 3.440 \quad \text{and} \quad d_3 = \frac{\sqrt{238357395880861}}{12252240} \cong 1.260.$$

Table 1 compares values for d_2 and d_3, given the sample is drawn from exponential, normal, Rayleigh, and $U(0,1)$ distributions for sample sizes $n = 2$ to $n = 20$. These constants do not depend on the rate parameter λ (for the exponential and Rayleigh distributions) nor μ or σ (for the normal distribution).

	d_2				d_3			
n	Expon	Normal	Rayleigh	$U(0,1)$	Expon	Normal	Rayleigh	$U(0,1)$
2	1.000	1.128	1.121	1.155	1.000	0.853	0.863	0.816
3	1.500	1.693	1.681	1.732	1.118	0.888	0.897	0.775
4	1.833	2.059	2.041	2.078	1.167	0.880	0.885	0.693
5	2.083	2.326	2.300	2.309	1.193	0.864	0.866	0.617
6	2.283	2.534	2.501	2.474	1.210	0.848	0.848	0.553
7	2.450	2.704	2.663	2.598	1.221	0.833	0.830	0.500
8	2.593	2.847	2.797	2.694	1.230	0.820	0.815	0.455
9	2.718	2.970	2.912	2.771	1.235	0.808	0.802	0.418
10	2.829	3.078	3.012	2.834	1.241	0.797	0.790	0.386
11	2.929	3.173	3.100	2.887	1.245	0.787	0.779	0.358
12	3.020	3.258	3.179	2.931	1.248	0.778	0.769	0.334
13	3.103	3.336	3.250	2.969	1.251	0.770	0.759	0.313
14	3.180	3.407	3.314	3.002	1.253	0.762	0.752	0.294
15	3.252	3.472	3.373	3.031	1.255	0.755	0.745	0.278
16	3.318	3.532	3.427	3.057	1.257	0.749	0.738	0.263
17	3.381	3.588	3.477	3.079	1.259	0.743	0.731	0.250
18	3.440	3.640	3.524	3.099	1.260	0.738	0.726	0.238
19	3.495	3.689	3.568	3.118	1.261	0.733	0.720	0.227
20	3.548	3.735	3.608	3.134	1.263	0.729	0.715	0.217

Table 4.1: Comparison of d_2 and d_3 for exponential, normal, Rayleigh, and $U(0,1)$ sampling distributions.

As shown in Table 1, APPL is able to calculate exact values of d_2 and d_3 for the exponential, Rayleigh, and standard uniform distributions. All other distributions required numerical integration to calculate d_2 and d_3. So, in theory, we could estimate d_2 and d_3 for any arbitrary sampling distribution. While this might be novel, it is not special to APPL because we are really using Maple's capability to estimate the result with numerical integration. If we do provide numeric values for parameters, we can take advantage of APPL to calculate the constants. In some cases, as illustrated in Example 2, APPL provides exact results.

There may be applications (e.g., life testing associated with bulbs or fuses) where a non-normal distribution is appropriate, and this provides an easy way to calculate control chart constants. Additionally, Tadikamalla et al. (2008) substantiate non-normal applications providing examples that calculate the upper and lower control limits for the logistic and Laplace distributions. Though they only consider symmetric distributions, the same practice can be considered for nonsymmetric cases using APPL, with an added advantage of never referring to a chart calculated for specific values of n and kurtosis estimates.

Example 2. Given that X_1 and X_2 are iid Weibull(2, 3) random variables, find the bias correction factor d_2 for the sample range. The APPL statements

```
> n  := 2:
> X  := WeibullRV(2, 3):
> d2 := (Mean(OrderStat(X, n, n)) - Mean(OrderStat(X, n, 1)))
         / sqrt(Variance(X)):
```

yield

$$d_2 = \frac{\pi\sqrt{6}\left[\left(4 - 2^{2/3}\right)/18 - 2^{2/3}\sqrt{3}/6\right]}{\sqrt{9\Gamma(2/3)^3 - 2\pi^2}} \cong 1.135.$$

The APPL procedure `OrderStat(X, n, r)` computes the exact distribution of the r^{th} order statistic drawn from a sample of size n drawn from a population described by the random variable X.

In order to find d_2 and d_3 from first principles (as provided by Wheeler (2000)), given an underlying parametric distribution, we must assign values to the distribution parameters. Even with small sample sizes, the process control literature provides well-established parameter estimation methods. However, given the normal distribution's wide acceptance in process control, current literature focuses on the normal distribution's mean μ and standard deviation σ, potentially suggesting an area of further work. Conceivably, if we knew enough about the observed process data to use a non-normal parametric model, we should also be confident in estimating the distribution's parameters. Thus APPL provides an efficient foundation for calculating d_2 and d_3.

Selecting a distribution to adequately model observed data has many troubling issues. If the researcher does not want to make assumptions accompanying a certain parametric distribution nor introduce potential error in selection, he or she can also create a distribution via bootstrapping with well-established statistical properties (Efron and Tibshirani, 1993). Once a probability distribution function is created using bootstrapping, APPL can compute the constants d_2 and d_3 as shown in Example 3 using the `BootstrapRV` procedure.

Example 3. Given the arbitrary probability distribution function $f_X(x)$ created by bootstrapping for the observed order statistics $x_{(1)} = 1$, $x_{(2)} = 3$, $x_{(3)} = 4$, and $x_{(4)} = 7$, compute the constants d_2 and d_3 for sample size $n = 3$. The APPL statements

```
> data := [1, 3, 4, 7]:
> X    := BootstrapRV(data):
> R    := RangeStat(X, 3):
> d2   := Mean(R) / sqrt(Variance(X));
> d3   := sqrt(Variance(R)) / sqrt(Variance(X));
```

yield

$$d_2 = 19\sqrt{3}/20 \cong 1.645 \qquad \text{and} \qquad d_3 = \sqrt{2637}/60 \cong 0.856.$$

4.3 Constants c_4, c_5

Similar to d_2 and d_3, the control chart constants c_4 and c_5 are also bias correction factors. However, as d_2 and d_3 corrected for the mean and standard deviation of the sample range R, c_4 and c_5 correct for the mean of the sample standard deviation, S, and its standard error. This is unusual because we usually discuss a sample's mean and standard deviation, but we are now focused on the *sample's mean standard deviation* and *the variance of the standard deviation.* We denote the mean of the standard deviation by μ_S and its standard deviation by σ_S. Thus the relationships are

$$\mu_S = E\left[S\right] = c_4 \sigma_X \tag{4.5}$$

and

$$\sigma_S = \sqrt{\mathrm{Var}(S)} = c_5 \sigma_X. \tag{4.6}$$

4.3.1 Normal Sampling

The derivations of c_4 and c_5 are based on the fact that $E\left[S^2\right] = \sigma_X^2$ and the well-known result

$$\frac{(n-1)S^2}{\sigma_X^2} \sim \chi_{n-1}^2 \tag{4.7}$$

for normal sampling (Hogg et al., 2005), where χ_{n-1}^2 denotes a chi square random variable with $n-1$ degrees of freedom. The mean of the sample standard deviation is

$$\begin{aligned}
c_4\sigma_X &= E\left[S\right] \\
&= E\left[\sqrt{S^2}\right] \\
&= E\left[\sqrt{S^2\frac{n-1}{n-1}\cdot\frac{\sigma_X^2}{\sigma_X^2}}\right] \\
&= E\left[\frac{\sigma_X}{\sqrt{n-1}}\sqrt{\frac{(n-1)S^2}{\sigma_X^2}}\right] \\
&= \frac{\sigma_X}{\sqrt{n-1}}\,E\left[\sqrt{\chi^2_{n-1}}\right].
\end{aligned}$$

Solving for c_4 yields

$$c_4 = \frac{E\left[\chi_{n-1}\right]}{\sqrt{n-1}},$$

where χ_{n-1} denotes a chi random variable with $n-1$ degrees of freedom. The standard deviation of the sample standard deviation is

$$\begin{aligned}
c_5\sigma_X &= \sigma_S \\
&= \sqrt{\text{Var}\left[S\right]} \\
&= \sqrt{E\left[S^2\right] - \left[E\left[S\right]\right]^2} \\
&= \sqrt{\sigma_X^2 - E\left[S\right]E\left[S\right]} \\
&= \sqrt{\sigma_X^2 - \frac{\sigma_X^2}{n-1}E\left[\sqrt{\chi^2_{n-1}}\right]^2} \\
&= \sigma_X\sqrt{1-\frac{E\left[\chi_{n-1}\right]^2}{n-1}}.
\end{aligned}$$

Solving for c_5 yields

$$c_5 = \sqrt{1-\frac{E\left[\chi_{n-1}\right]^2}{n-1}}.$$

The result provided in (4.7) yields a distinct advantage for finding c_4 and c_5 in the normal sampling case. We can use APPL to perform the calculations independently of the parameters σ and μ, producing the exact results for c_4 and c_5 which depend only on the sample size n. The procedure `c4(n)` is given below. A similar procedure, c5(n), was written for c_5.

```
> c4 := proc(n)
>   local X, c4;
>   X := ChiRV(n - 1):
>   c4 := Mean(X) / sqrt(n - 1):
>   return(c4);
> end proc;
```

A call to c4 and c5 with the argument $n = 4$, for example, yields the exact values

$$c_4 = \frac{2\sqrt{6}}{3\sqrt{\pi}} \cong 0.921 \qquad \text{and} \qquad c_5 = \frac{1}{3}\sqrt{9 - \frac{24}{\pi}} \cong 0.389.$$

These symbolic expressions are somewhat novel in that these constants are typically tabulated in decimal form rather than exactly in symbolic form. Furthermore, to illustrate the value of the APPL application and Maple's symbolic computational ability, consider the unlikely large sample size $n = 100$. A call to c4(100) produces

$$c_4 = \frac{39614081257132168796771975168\sqrt{22}}{105095150568296034723763017975\sqrt{\pi}} \cong 0.997.$$

The associated exact expression for c_5 is much too large to fit here, but the numerical value is $c_5 \cong 0.071$. The CPU time to compute these constants is negligible.

4.3.2 Non-Normal Sampling

Given that observations $X_1, X_2, \ldots, X_n$, are sampled from a non-normal distribution calculating c_4 and c_5 is much more complicated. We first derive a general form of each, then investigate its calculation for select distributions.

Using (4.5), the general derivation of c_4 is

$$\begin{aligned}
c_4\sigma_X &= E\left[S\right] \\
&= E\left[\sqrt{S^2}\right] \\
&= E\left[\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(X_i-\bar{X}\right)^2}\right] \\
&= \frac{1}{\sqrt{n-1}}\,E\left[\sqrt{\sum_{i=1}^{n}X_i^2-2n\bar{X}^2+n\bar{X}^2}\right] \\
&= \frac{1}{\sqrt{n-1}}\,E\left[\sqrt{\sum_{i=1}^{n}X_i^2-n\bar{X}^2}\right] \\
&= \frac{1}{\sqrt{n-1}}\,E\left[\sqrt{\sum_{i=1}^{n}X_i^2-\left[\sum_{i=1}^{n}X_i\right]^2\Big/n}\right].
\end{aligned}$$

Therefore, we calculate c_4 as

$$c_4=\sigma_X^{-1}\left[\frac{1}{\sqrt{n-1}}\,E\left[\sqrt{\sum_{i=1}^{n}X_i^2-\left[\sum_{i=1}^{n}X_i\right]^2\Big/n}\right]\right]. \tag{4.8}$$

In a similar manner, and using (4.6), it can be shown that a general expression for c_5 is

$$c_5=\sigma_X^{-1}\sqrt{\sigma_X^2-\frac{1}{n-1}\left(E\left[\sqrt{\sum_{i=1}^{n}X_i^2-\left(\sum_{i=1}^{n}X_i\right)^2\Big/n}\right]\right)^2}.$$

Burr (1976) also presents c_5 in terms of c_4 via the relationship

$$c_5=\sqrt{1-c_4^2}.$$

Therefore, if we are successful in finding c_4 we can easily calculate c_5, narrowing the focus of evaluation to c_4. Substituting $n=2$ into (4.8), we conclude that the numerator, $E(S)$, is

$$E[S] = \frac{1}{\sqrt{2}} E[\,|X_1 - X_2|\,].$$

The bias correction factor is then calculated via

$$c_4 = E[S]/\sigma_X = \frac{1}{\sqrt{2}\sigma_X} E[\,|X_1 - X_2|\,].$$

Given that the parameter σ_X appears in the denominator of the expression, we require it to also appear in the numerator forcing a cancellation and a numerical c_4 value that is independent of σ_X. Unfortunately, this only occurs for distributions where a single parameter involving the standard deviation appears. The next example highlights such an occurrence.

Example 3. Given that X_1, and X_2 are iid exponential(λ) random variables, find the bias correction factor c_4 for the sample standard deviation. The APPL statements

```
> X := ExponentialRV(lambda):
> Y := Difference(X, X):
> g := [[x -> -x, x -> x], [-infinity, 0, infinity]]:
> Z := Transform(Y, g):
> Mean(Z) / sqrt(2 * Variance(X)):
```

yield $c_4 = \sqrt{2}/2 \cong 0.707$.

APPL also successfully executes the same code for $n = 2$ for the normal distribution ($c_4 = \sqrt{2/\pi} \cong 0.798$, which matches the $n = 2$ tabulated value exactly), exponential distribution ($c_4 = \sqrt{2}/2 \cong 0.707$), Erlang distribution ($c_4 = 3/4$), hyperbolic secant distribution ($c_4 \cong 0.768$), Rayleigh distribution ($c_4 = \frac{\sqrt{2\pi}-\sqrt{\pi}}{\sqrt{4-\pi}} \cong 0.792$), and the $U(0,1)$ distribution ($c_4 = \sqrt{6}/3 \cong 0.816$).

When $n = 3$, the mean of the sample standard deviation is

$$E[S] = \frac{1}{\sqrt{6}}\,E\left[\sqrt{2X_1^2 + 2X_2^2 + 2X_3^2 - 2X_1X_2 - 2X_1X_3 - 2X_2X_3}\right].$$

The appearance of the random variables X_1, X_2, and X_3 at various positions in the expected value expression make the evaluation of $E[S]$ more difficult. Monte Carlo simulation must be relied on to provide the bias correction factors c_4 and c_5. Table 2 provides estimates of c_4 and c_5 using ten million replications (which ensures that the factors are accurate to three digits after the decimal point) for the same distributions considered in Table 1. The $n = 2$ row and normal columns are consistent with the exact results provided by APPL.

	c_4				c_5			
n	Expon	Normal	Rayleigh	$U(0,1)$	Expon	Normal	Rayleigh	$U(0,1)$
2	0.707	0.798	0.792	0.816	0.707	0.602	0.610	0.577
3	0.797	0.886	0.882	0.912	0.604	0.463	0.472	0.410
4	0.839	0.921	0.917	0.946	0.544	0.389	0.398	0.324
5	0.865	0.938	0.935	0.962	0.501	0.346	0.354	0.272
6	0.883	0.949	0.948	0.972	0.469	0.314	0.318	0.237
7	0.897	0.957	0.956	0.977	0.443	0.289	0.294	0.212
8	0.907	0.963	0.964	0.981	0.420	0.270	0.267	0.194
9	0.916	0.967	0.967	0.984	0.401	0.254	0.254	0.180
10	0.923	0.971	0.969	0.986	0.386	0.240	0.245	0.169

Table 4.2: Values of c_4 and c_5 for exponential, normal, Rayleigh, and $U(0,1)$ sampling distributions obtained by Monte Carlo simulation.

4.4 Conclusions

The control chart constants d_2, d_3, c_4, and c_5 can be calculated symbolically using a computer algebra system in the case of sampling from a normal population. In addition, d_2 and d_3 can be calculated symbolically for several non-normal populations and c_4 and c_5 can be calculated symbolically for several non-normal populations when $n = 2$. These calculations were performed with the aid of the Maple–based APPL software, which is available at no cost to non-commercial users at `www.APPLSoftware.com`. Monte Carlo simulation can be used to estimate control chart constants that can not be calculated symbolically.

Chapter 5

Testing Conformance to Benford's Law

5.1 Introduction

Frank Benford published "The Law of Anomalous Numbers" in 1938 in which he gathered over 20,000 data values from various fields (Benford, 1938). He correctly concluded the more general probability law suggesting that leading digits are not uniformly distributed over the natural numbers $1, 2, \ldots, 9$. Simon Newcomb (1881) made a similar observation more than fifty years earlier in his 1881 article on the frequency of use in logarithm tables. He noted that the earlier pages in a book of common logarithm tables were more worn than the pages at the end of the book, suggesting these pages were referenced more frequently. Though both observations occurred more than fifty years apart, the authors' conclusions are amazingly similar, with Benford capturing most of the credit for the logarithmic phenomenon known today as Benford's law (Hill, 1996). In addition to the attention given to the distribution of the first digit, Benford's law follows-up with a distribution of the second digit through the qth digit as well as their joint distribution. His case study in a wide range of applications (e.g., population, physics, voltage, addresses) lent additional credibility to the probabilistic logarithmic relation

$$\Pr(X = x) = \log_{10}(1 + 1/x)\,,$$

for $x = 1, 2, \ldots, 9$ (Larsen and Marx, 2006). We will refer to this as the Benford distribution. The wide range of applications of Benford's law includes the one-day return on stock market indexes (Ley, 1996), detecting accounting fraud (Nigrini, 1996), the distribution of the population of 3,141 counties in the 1990 U.S. Census (Nigrini and Wood, 1995), and election forensics (Mebane, 2006). It is the accepted method of testing data for human influence since such influence typically interrupts the naturally occurring distribution of first significant digits. We refer to the first significant digit as the first non-zero digit in a number (e.g., the first significant digit of 213 is 2 and the first significant digit of 0.00143 is 1). In addition, results concerning scale-invariance (Pinkham, 1961), base-invariance (Hill, 1995), and mixtures (Rodriguez, 2004) potentially offer even more utility in applying Benford's law. Benford and Nigrini suggest that data conforming to Benford's law satisfy the following conditions. (a) The data must be numeric (and not categorical) because the Benford distribution represents the frequencies of leading digits in numerical data sets. (b) The data must share a relation to the same phenomenon (e.g., residential addresses). Nigrini suggests, for example, stock prices are influenced by competing economic and financial forces. (c) The data must not be restricted by minimum or maximum values thus restricting the support of possible values the random variable of interest might assume. (d) The data must occur naturally (without human influence or bias), and they are not invented nor assigned, such as telephone numbers or social security numbers. Since these numbers can be allocated in any predetermined order, the distribution of leading digits in assigned numbers could be biased toward certain digits. (e) The data must contain at least four digits.

This chapter suggests the use of the Kolmogorov–Smirnov (KS) test over the more traditional chi-square goodness-of-fit (GOF) test for assessing Benford's law. The KS test accommodates small sample sizes and is exact under the null hypothesis. The performance of the two tests are compared for several alternatives.

5.2 Traditional Conformance Testing

An accepted method for testing conformance to Benford's law is the Pearson chi-squared GOF test. Let X be a random variable having the Benford distribution. For the continuous random variable T with cumulative distribution function $F_T(t)$ and associated *iid* observations $t_1, t_2, \ldots, t_n$, let the random variable Y be the leading digit in T and let $y_1, y_2, \ldots, y_9$ be the tallys of the leading digits. Thus, the probability mass function for Y is (Leemis, et al., 2000)

$$\Pr(Y = y) = \sum_{i=-\infty}^{\infty} \left[F_T(y \cdot 10^i) - F_T((y-1) \cdot 10^i)\right]$$

for $y = 1, 2, \ldots, 9$. The null and alternative hypotheses for the test are

H_0: the random variable Y has the Benford distribution,

H_a: the random variable Y does not have the Benford distribution.

The chi-square goodness-of-fit test statistic for this test using the Benford probabilities is

$$c = \sum_{i=1}^{9} \frac{\left[y_i - n\log_{10}(1+1/i)\right]^2}{n\log_{10}(1+1/i)}.$$

Letting $p_i = \log_{10}(1+1/i)$ for $i = 1, 2, \ldots, 9$ the expression for the test statistic is

$$c = \sum_{i=1}^{9} \frac{(y_i - np_i)^2}{np_i}.$$

The distribution of the chi-square statistic c is approximately χ^2 with eight degrees of freedom under H_0 when the expected number of observations in each cell exceeds five (i.e., $n\log_{10}(1+1/9) = 0.0457n > 5 \Rightarrow n > \frac{5}{0.0457} = 109$). The test measures the discrepancy between the observed cell frequency and the expected cell frequency. The closer match between the observed and expected frequencies, the more plausible is the null hypothesis and vice versa. This test rejects the null hypothesis at significance level α if the test statistic exceeds $\chi^2_{8,\alpha}$, where α is a right-hand tail probability.

Example 1. Consider the continuous probability density function

$$f_T(t) = \frac{1}{t \ln 10} \qquad 1 < t < 10$$

which satisfies Benford's law exactly (for a detailed explanation see Leemis, et al., 2000). Using this distribution, we conduct a Monte Carlo simulation in which samples are generated from this distribution and then tested as described above. We arbitrarily set the significance level to $\alpha = 0.05$ and use sample sizes $n = 25, 50,$ and 100. The simulation tracks the fraction of time the null hypothesis is rejected in 500,000 replications. For each sample size, a confidence interval for the fraction of rejections is calculated as described by Leemis and Trivedi (1996). Since the test is asymptotically exact and the sample size $n = 25$ does not meet the cell requirement of at least five observations per cell we expect the resulting confidence interval coverage to differ from the nominal five percent. For $n = 50$ and 100, even though the $n \geq 109$ requirement is not met, it appears that the results are more reliable. We combine the results of the simulation with those for the next distribution below in Table 5.1.

Now let the random variable $W \sim U(0, 2)$. Furthermore, let $V = 10^W$. Then the probability density function of V is

$$f_V(v) = \frac{1}{2v \ln 10} \qquad 1 < v < 100.$$

If Y is the leading digit of V it can be shown that the probability mass function of Y has the Benford distribution (Leemis, et al., 2000). We proceed as in the previous distribution for the random variable V. Table 5.1 depicts confidence intervals for the fraction of rejections under H_0 for the various sample sizes. Only $n = 100$ produces intervals that cover the desired value 0.05 for both T and V.

Distribution	Fraction rejected under H_0		
	$n = 25$	$n = 50$	$n = 100$
$f_T(t)$	$(0.0513, 0.0525)$	$(0.0502, 0.0514)$	$(0.0499, 0.0510)$
$f_V(v)$	$(0.0516, 0.0528)$	$(0.0500, 0.0512)$	$(0.0496, 0.0508)$

Table 5.1: Confidence intervals ($\alpha = 0.05$) for the fraction of tests rejected in 500,000 replications.

The confidence intervals in Table 5.1 depict the chi-square GOF test's dependence on sample size. This is problematic for the test for small sample sizes. We note the poor performance for $n = 25$ and $n = 50$ at capturing $\alpha = 0.05$ despite the high number of replications conducted in the simulation. One additional shortcoming in both contrived examples involves the lower limit of the random variable's support. Typically the more general case for a random variable, say T, is desired such as $t > 0$. To capture the added requirement $0 < t < 1$ we introduce the integer D such that D satisfies

$$10^D \leq T \leq 10^{D+1},$$

where T is a continuous random variable with positive support and such that the leading digit satisfies Benford's law. Using this notation we can capture the leading digit of T for any magnitude of D, where $-\infty < D < \infty$.

Using the chi-squared GOF test statistic c as defined above, we can calculate the exact distribution of the test statistic by enumerating the 9^n possible outcomes for a sample size n. The simplest distribution occurs when $n = 2$. For a sample of size

$n = 2$ the generalized probability distribution function is

$$f_C(c) = \begin{cases} p_i^2 & c = \frac{(2-2p_i)^2}{2p_i} + \sum_{j \neq i} 2p_j \\ 2p_i p_j & c = \frac{(1-2p_i)^2}{2p_i} + \frac{(1-2p_j)^2}{2p_j} + \sum_{k \neq i,j} 2p_k \end{cases}$$

for $i, j, k = 1, 2, \ldots, 9$. Using each of the $9^2 = 81$ outcomes, the probability of rejecting H_0 can be calculated exactly for a given sample size n by comparing c to $\chi^2_{8,\alpha}$. Table 5.2 provides the exact probabilities of rejecting H_0 at $\alpha = 0.05$ for $n = 2$ up to $n = 12$ and Monte Carlo estimates (due to CPU limitations) when $n > 12$.

n	2	3	4	5	6	7	8
Pr (reject H_0)	0.050	0.059	0.071	0.068	0.056	0.055	0.055

n	9	10	11	12	13	14	15
Pr (reject H_0)	0.058	0.057	0.057	0.056	0.054	0.053	0.053

Table 5.2: Probability of rejecting H_0 under H_0.

5.3 Alternative Method for Conformance Testing

For the probability distribution function (PDF) given in Example 1, the cumulative distribution function (CDF) is

$$F_T(t) = \frac{\ln t}{\ln 10} \qquad 1 < t < 10$$

and the associated variate generation algorithm via inversion of the CDF is

$$T \leftarrow 10^U$$

where $U \sim U(0, 1)$. We now define a new random variable Z as

$$Z = \log_{10} T \bmod 1.$$

For this new random variable Z, using the substitution for T in the variate generation algorithm

$$\begin{aligned} Z &= \log_{10} T \bmod 1 \\ &= \log_{10}(10^U) \bmod 1 \\ &= U \bmod 1 \\ &= U \end{aligned}$$

which is $U(0,1)$. Thus $Z \sim U(0,1)$. This suggests that testing whether the leading digit of this distribution conforms to Benford's distribution is equivalent to testing whether $Z \sim U(0,1)$. A more detailed example follows.

Example 2. Let the continuous random variable W have the piecewise pdf

$$f_W(w) = \begin{cases} w & 0 < w < 1 \\ \frac{2-w}{4} & 1 < w < 2 \\ \frac{9-3w}{4} & 2 < w < 3, \end{cases}$$

and consider the transformation $T = 10^W$. The resulting pdf for T is

$$f_T(t) = \begin{cases} \frac{\ln t}{t \ln(10)^2} & 1 < t < 10 \\ \frac{1}{2t \ln 10} - \frac{\ln t}{4t \ln(10)^2} & 10 < t < 100 \\ \frac{9}{4t \ln 10} - \frac{3 \ln t}{4t \ln(10)^2} & 100 < t < 1000, \end{cases}$$

which is known to satisfy Benford's law exactly (Leemis, et al., 2000). The cdf of $Z = \log_{10} T - D$, with $W = \log_{10} T$ from the transformation above, is

$$\begin{aligned}
F_Z(z) &= \Pr(Z \le z) \\
&= \sum_{d=-\infty}^{\infty} \Pr(10^d \le T < 10^{d+1}) \cdot \Pr(\log_{10} T - d \le z | 10^d \le T < 10^{d+1}) \\
&= \sum_{d=0}^{2} \Pr(d \le W < d+1) \cdot \Pr(W - d \le z | d \le W < d+1) \\
&= \frac{1}{2} \cdot \Pr(W \le z | 0 \le W < 1) + \frac{1}{8} \cdot \Pr(W - 1 \le z | 1 \le W < 2) + \\
&\qquad \frac{3}{8} \cdot \Pr(W - 2 \le z | 2 \le W < 3) \\
&= \frac{1}{2} \int_0^z 2y\, dy + \frac{1}{8} \int_0^z (2 - 2y) dy + \frac{3}{8} \int_0^z (2 - 2y) dy \\
&= \frac{z^2}{2} + \frac{2z - z^2}{8} + \frac{6z - 3z^2}{8} \\
&= z
\end{aligned}$$

for $0 < z < 1$. Thus $Z \sim U(0, 1)$. The geometry associated with the pdf of W is shown in Figure 5.1. The solid lines are the PDF of W and the dashed lines are the segments outside of the range $[0, 1]$ translated to $[0, 1]$, along with the resultant sum. This remarkable result shows that if the segments of the pdf are translated to $0 < w < 1$ and sum to unity, then Benford's law is satisfied exactly. This generalizes in the following theorem.

Theorem 5.1 *If T is a continuous random variable with support that is a subset of $(0, \infty)$ and $\log_{10} T \bmod 1 \sim U(0, 1)$, then the leading digit of T has the Benford distribution.*

Proof Let $W = \log_{10} T$ and D as defined earlier. Substituting W for $\log_{10} T$ results in $W \bmod 1 \sim U(0, 1)$. The mod operation effectively removes the quantity left of the decimal point in W. This is equivalent to shifting W left as shown in Figure 5.1 of Example 2. This shifting can also be characterized as subtracting the order of

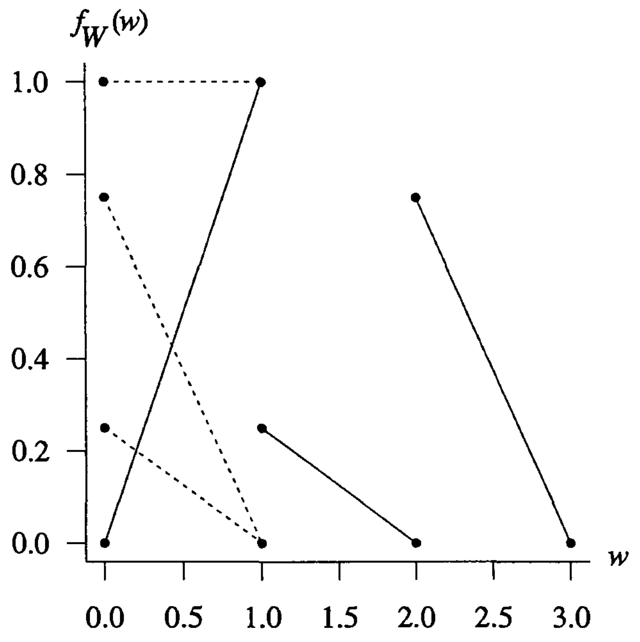


Figure 5.1: Geometry associated with Example 3.

magnitude $W - D$. This also removes the digits in W left of the decimal place, which establishes the support of $Z = \log_{10} T - D$ as $[0, 1]$. Consider the leading digit, $Y = y$. Summing over all possible orders of magnitude D yields

$$\begin{aligned}
\Pr(Y=y) &= \sum_{d=-\infty}^{\infty} \Pr\left(10^d y \le T < 10^d(y+1)\right) \\
&= \sum_{d=-\infty}^{\infty} \Pr\left(\log_{10}(10^d y) \le \log_{10}(T) < \log_{10}(10^d(y+1))\right) \\
&= \sum_{d=-\infty}^{\infty} \Pr\left(d+\log_{10}(y) \le \log_{10}(T) < d+\log_{10}(y+1)\right) \\
&= \sum_{d=-\infty}^{\infty} \Pr\left(\log_{10}(y) \le \log_{10}(T)-d < \log_{10}(y+1)\right) \\
&= \sum_{d=-\infty}^{\infty} \Pr\left(\log_{10}(y) \le Z < \log_{10}(y+1)\right) \\
&= F_Z(\log_{10}(y+1)) - F_Z(\log_{10}(y)) \\
&= \log_{10}(y+1) - \log_{10}(y) \\
&= \log_{10}\frac{y+1}{y} \\
&= \log_{10}(1+1/y),
\end{aligned}$$

for $y = 1, 2, \ldots, 9$, which is the probability mass function for the Benford distribution.

■

Though it would be desirable for Theorem 5.1 to be *if and only if*, the converse of the theorem is not true. Consider the following counter-example. Let the continuous random variable T have pdf

$$f_T(t) = \log_{10}(1+1/i), \qquad i \le t < i+1; \qquad i = 1, 2, \ldots, 9.$$

This conforms exactly to the Benford distribution where the support is limited to only the first order of magnitude, making the subtraction of D unnecessary. Using the transformation technique, the distribution of $W = \log_{10} T$ is

$$f_W(w) = \log_{10}\left(\frac{i+1}{i}\right)\cdot \ln(10)\cdot 10^w, \qquad \log_{10}(i) \le w < \log_{10}(i+1); \qquad i = 1, 2, \ldots, 9$$

which is clearly not uniformly distributed on the interval $(0, 1)$. Therefore, Theorem 5.1 applies for a specific class of Benford populations as illustrated in Examples 1 and 2.

5.4 Testing via Kolmogorov–Smirnov

The result from Theorem 5.1 allows use of the Kolmogorov–Smirnov (KS) test. Under the null hypothesis, testing for conformance to the Benford distribution is equivalent to testing $\log_{10} T \bmod 1$ against the standard uniform distribution. There are two immediate benefits arising from this alternate test, (a) the KS test is exact and (b) the KS test is appropriate for small sample sizes (the rule of thumb required $n > 109$ for the chi-square GOF test). The results for Table 5.2 were extended to $n = 40$ for $\alpha = 0.05$ and $\alpha = 0.01$ in Figure 5.2. The same probabilities were calculated for values up to $n = 120$, however, the behavior is as expected for $n > 40$, thus we chose $n = 40$ as the upper limit on the plot for clarity. As depicted, the KS test is exact for $n \geq 1$, providing superior performance over the chi-square GOF test. As n approaches 109, the chi-square GOF test probability of rejecting H_0 is sufficiently close to the associated KS value. The stellar performance of the chi-square GOF test for $n = 2$ and $\alpha = 0.05$ is purely coincidental. It would also be of interest to compare the two techniques for mixtures of distributions that morph from exactly Benford to some non-Benford distribution associated with the alternative hypothesis. To test these instances, we first fix the sample size and significance level at $n = 50$ and $\alpha = 0.05$. We then plot the power curves for the two tests as the distribution morphs from a Benford population to some non-Benford distribution by introducing a biased coin flip variable, where with probability p a non-Benford variate is produced and with probability $1 - p$ a Benford variate is produced. The chosen non-Benford distributions, all with support on $1 \leq t < 10$, are (a) $U(1, 10)$ (b) anti-Benford (c) triangular$(1, 5.5, 10)$ and (d) inverted triangular. The anti-Benford distribution has pdf

$$f_T(t) = \frac{1}{(11-t)\ln(10)} \qquad 1 \le t < 10,$$

and the inverted triangular distribution has pdf

$$f_T(t) = \begin{cases} 22/81 - 4x/81 & 1 \le t < 5.5 \\ 4x/81 - 22/81 & 5.5 \le t < 10. \end{cases}$$

The Monte Carlo experiment consists of ten million replications for each value of p, the probability the distribution is other than Benford. We increment p by 0.01, providing 101 points for each power curve. For each replication the KS and chi-square GOF test statistics are compared to the associated critical values. The experiment returns the proportion of outcomes that reject the null hypotheses. Figure 5.3 provides side-by-side comparisons for each of the chosen distributions.

For the uniform and triangular distributions, the power curve of the KS test dom-

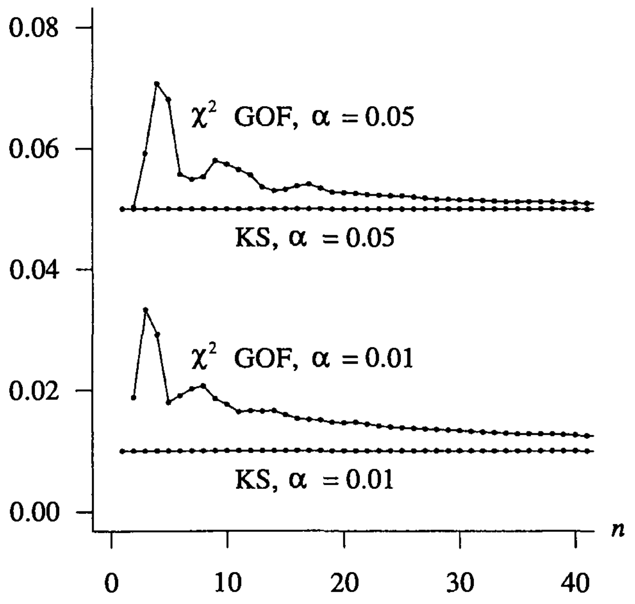


Figure 5.2: Probability of rejection under H_0 for the KS and chi-square GOF tests for various sample sizes.

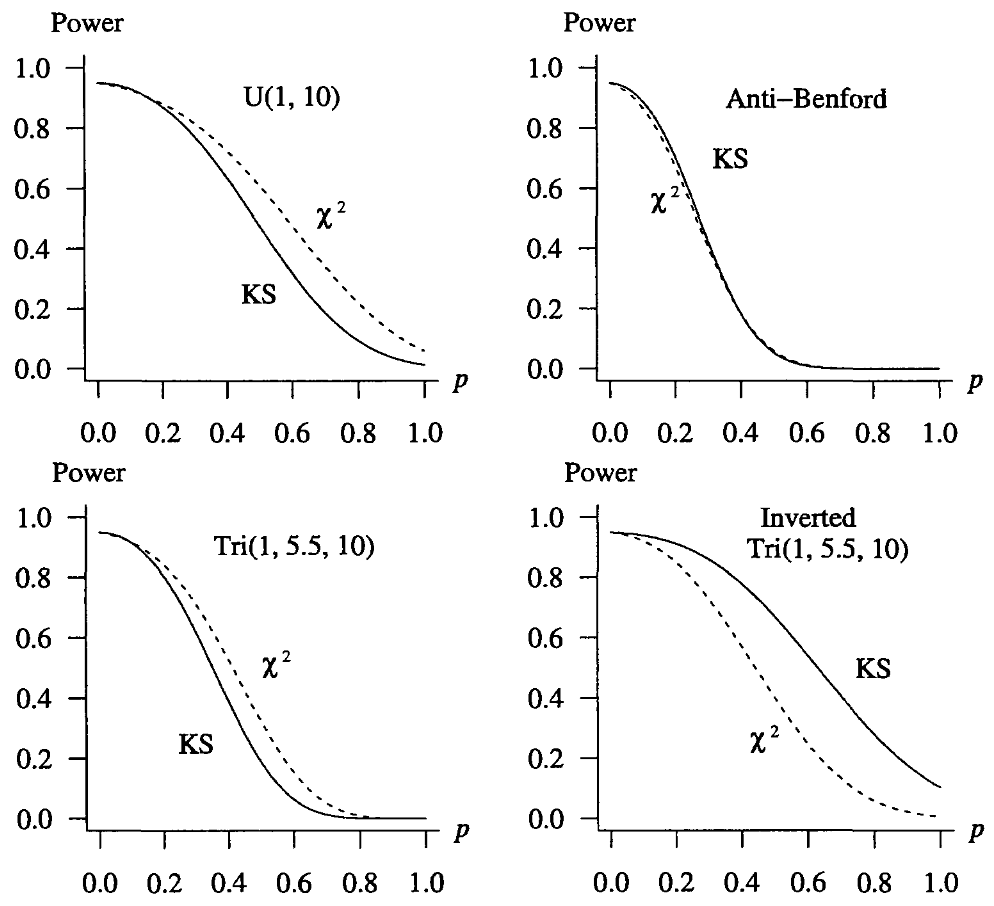


Figure 5.3: Power curves for the KS and chi-square GOF tests.

inates the chi-square GOF test. The anti-Benford distribution exhibits indifference when comparing the power curves and finally, the inverted triangular distribution favors the chi-square GOF test. Since the inverted triangular distribution occurs less frequently in practice than the others, we recommend the KS GOF test over the chi-square GOF test.

5.5 Conclusions

Due to the availability of diverse digital data, the opportunity for leading digit statistical testing is becoming more prevalent in government and industry. Thus Benford's law (especially the distribution of the leading digit) is being applied to many diverse circumstances in the current literature. The chi-squared GOF test is the current standard for checking conformance to Benford's law. Although this test is asymptotically exact, it requires a sufficiently large sample size before yielding reliable results. Additionally, for smaller sample sizes, the probability of rejecting the null hypothesis under H_0 can be erratic rather than monotonic with increasing sample size. An alternative test, the KS test, is appropriate and provides better performance as measured by power, exactness, and flexibility in sample size for the class of Benford populations where for the continuous random variable T, $\log_{10} T \bmod 1 \sim U(0,1)$. This test is easy to implement and offers the additional advantage of the ability to test small samples.

Chapter 6

Transient Queueing Analysis

6.1 Introduction

Many traditional simulation studies analyze queueing systems in steady-state, requiring appropriate warm-up periods and associated long simulation runs. However, in many cases the system being modeled never reaches steady-state; thus steady-state simulation results do not accurately portray the system behavior. The ability to analyze transient results associated with such models is often complicated by intractable theory, leaving simulation as the only method for analysis. Further complicating the transient analysis is the effect of initial conditions (Kelton and Law, 1985). Since steady-state results depend on running the system long enough to negate the impact of initial conditions, these steady-state results reveal nothing about the transient behavior of the queueing system. Our purpose here is to combine new and existing results in transient queueing analysis with a symbolic engine in computational probability.

There are many classes of queueing systems where a transient analysis is required, e.g., service businesses often model queues that never reach equilibrium. Recognizing the need to develop theory for transient results, as opposed to steady-state results, has resulted in a wide literature in this area. Initial work in transient analysis ironically appeared as an attempt to measure when a system achieved equilibrium. Law (1975) notes the consequences of failing to adequately account for the initial transient period, leading to Gafarian, et al. (1976) outlining a comprehensive framework for the initial transient problem. Morisaku (1976) addresses the time to equilibrium in simulations modeling the $M/M/1$ queue and provides schematics for the transition probabilities given $k \geq 0$ customers initially present at time $t = 0$. Pegden and Rosenshine (1982) provide a closed-form solution for the probability of exactly i arrivals and j servicings over a time horizon of length t in an $M/M/1$ queue starting empty and idle, allowing the calculation of certain performance measures for a specified time period. Odoni and Roth (1983) take an empirical approach to compare observed and predicted transient state queue length for the $M/M/1$ queue, noting that for small values of t the expected queue length is strongly influenced by initial conditions, and provide a good approximation for an upper bound of time to steady-state. Kelton and Law (1985) consider the $M/M/s$ ($s \geq 1$) queue and provide expressions to calculate the probabilities of having up to $n + k$ customers in the system upon the arrival of the nth customer, where k is the number of customers in the system at time $t = 0$. They then apply these calculations to a variety of measures of performance with implications to convergence on steady-state delays and offer methods for choosing queue initialization in simulation. Much of the work in this chapter is motivated by their results. Kelton (1985) extends the previous work by considering $M/E_m/1$ and $E_m/M/1$ queues. Parthasarathy (1987) provides a transient solution for the probability that there are n customers in the system at time t for an $M/M/1$ queue. Abate and Whitt (1988) use Laplace transformations to analyze some transient results of interest in the $M/M/1$ queue. Leguesdron, et al. (1993) provide transient probabilities for the $M/M/1$ queue by inverting the generating function of the uniformized Markov chain describing the $M/M/1$ process. In this chapter we will focus on the transient analysis of the $M/M/1$ and the more general $M/M/s$ queues, specifically on the distribution of the nth customer's sojourn time, which is the sum of the nth customer's delay time and service time.

The $M/M/s$ queue is defined in Section 6.2 for a positive integer s, and a method is given for calculating the probability distribution of the number of customers an arriving customer sees upon arrival to an $M/M/s$ queue. Section 6.3 describes how the sojourn time distribution is calculated for a given customer in an $M/M/s$ queue with k customers initially present in the system, $k \geq 0$. Section 6.4 includes examples using the implemented procedures to calculate exact sojourn time distributions, related measures of performance, and graphical illustrations for varying parameters such as traffic intensity and number of customers in the system. Section 6.5 offers two approaches for calculating the covariance and correlation among customers in an $M/M/1$ queue. Section 6.6 extends the covariance and correlation calculations by automating the process of finding the joint probability distribution function between two customers, and provides the exact covariance and correlation calculations for varying traffic intensities. Section 6.7 concludes the chapter by reviewing the content. Commented code is available in the appendices for all computations conducted here.

6.2 Basics of the $M/M/s$ Queue

The $M/M/s$ queue is governed by *iid* exponential interarrival times (the arrival stream is a Poisson process) with arrival rate λ, and *iid* exponential service times among s identical servers, each with service rate μ. The interarrival times and the service times are mutually independent. The traffic intensity of the system is $\rho = \lambda/s\mu$. The system consists of a single queue with customers waiting to be serviced by one of the identical s parallel servers. If an arriving customer finds at least one idle server, the customer immediately proceeds to service; otherwise the customer joins the single queue of those waiting for service in a first-come, first-served manner. To achieve classic steady-state results the traffic intensity must satisfy $\rho < 1$. This critical assumption is not required in transient analysis, described here, because the system of interest never reaches equilibrium.

Let $P_k(n,i)$ be the probability that upon the arrival of the nth customer there are i customers in the system including the nth customer (in queue or in service), given k customers are present at time $t = 0$. Using propositions provided by Kelton and Law (1985), reprinted here for completeness (proofs are available in the reference), and a recursion algorithm, $P_k(n,i)$ for $i = 1, 2, \ldots, n+k$ can be computed. Using these probabilities, it is possible to find the distribution of the sojourn time for the nth customer in an $M/M/s$ queue, given k customers are present at time $t = 0$. Proposition 1 addresses the case of no exits prior to the nth customer's arrival, given $k \geq 1$. Proposition 2 is identical to Proposition 1 except that the system is empty and idle at $t = 0$ (i.e., $k = 0$). Proposition 3 addresses the case that the first customer finds $i - 1$ other customers present for $k > 0$. Proposition 4 is the more general case that customer $n \geq 2$ finds i other customers present, given $k \geq 0$.

Proposition 1. *If $k \geq 1$, then for $n \geq 1$,*

$$P_k(n, k+n) = \begin{cases} [\rho/(\rho+1)]^n & \text{if } k \geq s \\ \rho^n / \prod_{j=1}^{n} [\rho + (k+j-1)/s] & \text{if } k+n \leq s \\ \rho^n / \left[(\rho+1)^{n-s+k} \prod_{j=1}^{s-k} [\rho + (k+j-1)/s]\right] & \text{if } k < s < k+n. \end{cases}$$

Proposition 2. *For $n \geq 1$,*

$$P_0(n,n) = \begin{cases} \rho^n / \prod_{j=1}^{n} [\rho + (j-1)\ s] & \text{if } n \leq s \\ \rho^n / \left[(\rho+1)^{n-s} \prod_{j=1}^{s} [\rho + (j-1)/s]\right] & \text{if } n > s. \end{cases}$$

Proposition 3. *If $k \geq 1$, then for $2 \leq i \leq k$,*

$$P_k(1,i) = \begin{cases} \{\rho/[\rho + (i-1)/s]\} \prod_{j=1}^{k-i+1} \{1 - \rho/[\rho + (k-j+1)/s]\} & \text{if } k \leq s \\ \rho/(\rho+1)^{k-i+2} & \text{if } k > s \text{ and } i > s \\ \{\rho/[(\rho+1)^{k-s+1}[\rho + (i-1)/s]]\} \cdot & \\ \quad \prod_{j=1}^{s-i} \{1 - \rho/[\rho + (s-j)/s]\} & \text{if } i \leq s < k. \end{cases}$$

Proposition 4. *For $n \geq 2$, and $2 \leq i \leq k+n-1$,*

$$P_k(n,i) = \begin{cases} [\rho/(\rho+1)] \sum_{j=i-1}^{k+n-1} [1/(\rho+1)]^{j-i+1} P_k(n-1,j) & \text{if} \quad i > s \\ \{\rho/[\rho+(i-1)/s]\} \cdot & \\ \quad \left\{\sum_{j=i-1}^{s-1} \left[\prod_{h=1}^{j-i+1} \{1-\rho/[\rho+(j-h+1)/s]\}\right] \cdot \right. & \\ \quad P_k(n-1,j) + \left[\prod_{h=1}^{s-i} \{1-\rho/[\rho+(s-h)/s]\}\right] \cdot & \\ \quad \left. \sum_{j=s}^{k+n-1} [1/(\rho+1)]^{j-s+1} P_k(n-1,j)\right\} & \text{if} \quad i \leq s. \end{cases}$$

Using these four propositions, $P_k(n,1)$ is calculated by subtracting the complementary probability from one. These results are coded in the Maple procedure `Queue(X, Y, n, k, s)`, where

- X is the exponential interarrival time distribution,
- Y is the exponential service time distribution,
- n is the index of the customer of interest,
- k is the number of customers in the system at time $t = 0$,
- s is the number of identical parallel servers.

The procedure is written in Maple and uses A Probability Programming Language (APPL), which can be downloaded for free at `www.APPLsoftware.com` and is described in Glen, et al. (2001). We choose to calculate the distribution of the sojourn time because it is a purely continuous random variable enabling us to exploit associated procedures in APPL. The `Queue` procedure and associated subprocedures are provided in Appendix D. The sojourn time distribution results provided by `Queue` were checked against a percentile comparison of $n = 10,000,000$ sojourn times created by the C code in Appendix E.

6.3 Creating the Sojourn Time Distribution

Once the necessary $P_k(n,i)$, $i = 1, 2, \ldots, n+k$, probabilities are calculated, the exact sojourn time distribution for the nth customer can be calculated. We define X_n as the number of customers, including customer n, in the system at time t, the arrival time of the nth customer. The possible values of X_n can vary from a minimum of 1, which occurs when customer n arrives to an empty queue, to a maximum of $n+k$, which occurs when 0 exits occur prior to customer n's arrival, matching the possible values for i in the expression $P_k(n,i)$ above. The mathematical derivations for both the $M/M/1$ and $M/M/s$ queues make extensive use of the memoryless property, permitting the construction of the distribution of T_n, the sojourn time of customer n. We present each case separately below.

6.3.1 Distribution of T_n for the $M/M/1$ Queue

For an $M/M/1$ queue starting empty and idle, the delay time of the first customer is zero because the customer proceeds directly to service upon arrival. Therefore, the first customer has an exponential(μ) sojourn time distribution. Conditioning on customer 1's service time, one can calculate the probabilities of customer 2 arriving before and after customer 1 finishes service. These well-known results (Kleinrock (1975), Hillier and Lieberman(2005), Winston (2004)) are

$$P(Y < X) = \frac{\mu}{\lambda + \mu}, \qquad P(X < Y) = \frac{\lambda}{\lambda + \mu}$$

where X is an exponential(λ) interarrival time and Y is an exponential(μ) service time. The first probability represents customer 2 proceeding directly to service, in which case his sojourn time is simply his service time, which is exponential(μ). The second probability represents the likelihood that customer 2 will delay prior to service. Using the memoryless property, customer 2 delays an exponential(μ) time before being serviced in an additional exponential(μ) time. Using these two probabilities, it is easy to see that customer 2's sojourn time distribution is a mixture, where the mix probabilities are the $P_0(n,i)$'s and the distributions are determined by the combinations of delays and services potentially encountered. It is well known that for $X_1, X_2, \ldots, X_n$ *iid* exponential(λ) random variables that

$$\sum_{i=1}^{n} X_i \sim \text{Erlang}(\lambda, n). \tag{6.1}$$

Using this result, the $M/M/1$ queue sojourn time distribution for $k = 0$ initial customers generalizes very elegantly to include $k > 0$, as indicated in Table 6.1. Line i of the table occurs with probability $P_k(n,i)$ and lists the distribution of the sojourn time for the nth customer, conditioned on i customers being in the system upon his arrival.

X_n	Delay	Service	Conditional sojourn time distribution
1	0	exponential(μ)	exponential(μ)
2	exponential(μ)	exponential(μ)	Erlang(μ, 2)
3	Erlang(μ, 2)	exponential(μ)	Erlang(μ, 3)
4	Erlang(μ, 3)	exponential(μ)	Erlang(μ, 4)
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n+k$	Erlang(μ, $n+k-1$)	exponential(μ)	Erlang(μ, $n+k$)

Table 6.1: Conditional sojourn time distributions for the $M/M/1$ queue.

Let $g_i(t)$ be the PDF of an Erlang(μ, i) random variable. Using the conditional sojourn time distributions for $i = 1, 2, \ldots, n+k$ potential customers in the system, each with probability $P_k(n,i)$, the PDF for the nth customer's sojourn time T_n is the mixture

$$f_n(t) = \sum_{i=1}^{n+k} P_k(n,i) g_i(t) \qquad t > 0. \tag{6.2}$$

This result is simple in the $M/M/1$ case because we can take advantage of (6.1), resulting in a mixture of $n + k$ Erlang distributions.

6.3.2 Distribution of T_n for the $M/M/s$ Queue

Given $s > 1$ parallel identical servers, the nth customer's sojourn time distribution is still a mixture of $n+k$ conditional sojourn time distributions. However, each distribution might be more complicated. For illustration, consider an $M/M/3$ queue starting empty and idle with exponential(λ) arrivals and three identical exponential(μ) servers. It is clear that for customers 1, 2, and 3, the sojourn time is exponential(μ) since all three customers proceed directly to service. Therefore, in the general case, for the number of customers in the system including customer n, which we defined as X_n, when $X_n \leq s$ the conditional sojourn time distribution is exponential(μ). However, if $X_n > s$, then the nth customer experiences a delay while observing $X_n - s$ service completions. When $s > 1$ and $X_n > s$, the service distribution observed by customers in queue is exponential with rate $s\mu$. Using this result, it is apparent that the delay time for the nth customer is the sum of $X_n - s$ independent exponential($s\mu$) random variables, and using (6.1) is Erlang($s\mu, X_n - s$). To calculate the nth customer's sojourn time for a particular value of X_n, we sum his delay time and his service time. Table 6.2 shows the distributions conditioned on the number of customers X_n encountered by customer n (including himself) for the $M/M/3$ queue, given $k = 0$ customers present at time $t = 0$. The APPL procedure `Convolution` calculates the distribution of a sum of independent random variables. We use the symbol $\oplus$ to represent convolution.

X_n	Delay	Service	Conditional sojourn time distribution
1	0	exponential(μ)	exponential(μ)
2	0	exponential(μ)	exponential(μ)
3	0	exponential(μ)	exponential(μ)
4	exponential(3μ)	exponential(μ)	exponential(3μ) $\oplus$ exponential(μ)
5	Erlang(3μ, 2)	exponential(μ)	Erlang(3μ, 2) $\oplus$ exponential(μ)
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	Erlang(3μ, $n-3$)	exponential(μ)	Erlang(3μ, $n-3$) $\oplus$ exponential(μ)

Table 6.2: Conditional sojourn time distributions for the $M/M/3$ queue with $k = 0$.

Since X_n represents the number of customers in the system upon arrival of the nth customer, including himself, the first row in Table 6.2 corresponds to customer n arriving to an empty system and the last row corresponds to no service completions prior to customer n's arrival. The general form for the $M/M/s$ sojourn time probability density function is identical to (6.2), however, in the $M/M/s$ case each $g_i(t)$ can potentially require an additional step to calculate the distribution of a sum of random variables.

6.4 Transient Analysis Applications

It is apparent that calculating (6.2) for large n is tedious. Kelton and Law (1985) acknowledge the computational difficulty in achieving the $P_k(n, i)$ probabilities alone. Conducting the added steps of up to $n-s$ convolutions for the $M/M/s$ queue and then mixing the resulting conditional distributions with the appropriate probabilities can be complicated to implement. APPL provides the underlying computational engine to achieve exact results for such problems. As mentioned earlier, the APPL procedure `Queue(X, Y, n, k, s)` returns the exact sojourn time distribution for customer n. `Queue` recursively calls `MMsQprob(n, k, s)`, which uses recursion to calculate the necessary $P_k(n, i)$ probabilities. APPL is capable of symbolic results, as illustrated in Examples 1 and 2.

Example 1. Consider an $M/M/1$ queue with arrival rate λ and service rate μ starting empty and idle at time $t = 0$. For the fourth customer, calculate the probabilities $P_0(4, i)$ for $i = 1, 2, 3, 4$.

The APPL command `MMsQprob(4, 0, 1)` returns the exact symbolic probabilities

$$\begin{aligned} P_0(4,1) &= \frac{5\rho^2+4\rho+1}{(\rho+1)^5} \\ P_0(4,2) &= \frac{\rho\,(5\rho^2+4\rho+1)}{(\rho+1)^5} \\ P_0(4,3) &= \frac{\rho^2\,(3\rho+1)}{(\rho+1)^4} \\ P_0(4,4) &= \frac{\rho^3}{(\rho+1)^3}, \end{aligned}$$

where $\rho = \lambda/\mu$. It is easy to verify that for any $\rho > 0$, $\sum_{i=1}^{4} P_0(4,i) = 1$, as required. For example, a simple substitution letting $\rho = 9/10$ yields

$$\begin{aligned} P_0(4,1) &= \frac{865000}{2476099} &\approx& \ 0.34933983 \\ P_0(4,2) &= \frac{778500}{2476099} &\approx& \ 0.31440585 \\ P_0(4,3) &= \frac{29970}{130321} &\approx& \ 0.22997061 \\ P_0(4,4) &= \frac{729}{6859} &\approx& \ 0.10628371. \end{aligned}$$

Example 2. For the queue described in Example 1, calculate the fourth customer's sojourn time distribution, mean sojourn time and sojourn time variance.

The APPL statements

```
X := ExponentialRV(lambda);
Y := ExponentialRV(mu);
T := Queue(X, Y, 4, 0, 1);
Mean(T);
Variance(T);
```

calculate the desired results. The first two lines define the interarrival and service time distributions, while the third line calculates the fourth customer's sojourn time distribution. The last two lines are self explanatory.

The resulting distribution is

$$\begin{aligned} f_4(t) &= \frac{1}{6(\lambda+\mu)^5}\mu^4 e^{-\mu t}\left(30\lambda^2 + 30\lambda^3 t + 24\lambda\mu + 24\lambda^2\mu t + 6\mu^2 + 6\mu^2\lambda t + \right. \\ &\qquad \left. 9t^2\lambda^4 + 12t^2\lambda^3\mu + 3t^2\lambda^2\mu^2 + t^3\lambda^5 + 2t^3\lambda^4\mu + t^3\lambda^3\mu^2\right) \quad t > 0. \end{aligned}$$

Using $f_4(t)$ above, the Mean and Variance commands return

$$E[T_4] = \frac{\mu^5 + 6\lambda\mu^4 + 26\mu^2\lambda^3 + 16\mu^3\lambda^2 + 17\mu\lambda^4 + 4\lambda^5}{\mu(\lambda+\mu)^5}$$

and

$$\begin{aligned} V[T_4] &= (181\mu^2\lambda^8 + 484\mu^3\lambda^7 + 816\mu^4\lambda^6 + 868\mu^5\lambda^5 + 574\mu^6\lambda^4 + \\ &\qquad 244\mu^7\lambda^3 + 40\mu\lambda^9 + 68\mu^8\lambda^2 + 12\mu^9\lambda + \mu^{10} + 4\lambda^{10}) \,/ \\ &\qquad \left(\mu^2(\lambda+\mu)^{10}\right). \end{aligned}$$

Substituting $\lambda = 1$ and $\mu = 10/9$, the results simplify to

$$f_4(t) = \frac{5000}{66854673}e^{-10/9t}\left(361t^3 + 2109t^2 + 5190t + 5190\right) \qquad t > 0,$$

$E[T_4] = \dfrac{23323347}{12380495} \approx 1.88387839$, and $V[T_4] = \dfrac{383506725720906}{153276656445025} \approx 2.50205566$.

The CPU time associated with the examples is negligible. Examples 1 and 2 represent simple applications of these procedures that circumvent time intensive hand-calculations. They serve only as indications of more challenging problems solvable using these procedures.

Example 3. Calculate the mean sojourn time of the 30th customer in an $M/M/2$ queue with arrival rate $\lambda = 1$, service rate $\mu = 9/20$ ($\rho = 10/9$), and $k = 3$ customers initially present.

The mean can be calculated in a single APPL statement by embedding the function calls

```
Mean(Queue(ExponentialRV(1), ExponentialRV(9/20), 30, 3, 2));
```

which yields

$$\frac{207470302076553093092838324788533105632236520526343624731399405569875101728767946601484880138641283564474794935548876340}{21534046672820071947860003352210296689224691678842510431455073374994143953948660661783359707587864512638771645692063053}$$

or, to 10 digits, 9.634524585.

Being able to represent the sojourn time distribution for the nth customer in closed form also provides valuable information on asymptotic behavior for queueing systems, including steady state convergence rates for different initial conditions. Figure 6.1 shows the mean sojourn time for customer $n = 1, 2, \ldots, 120$ in an $M/M/1$ queue with $\lambda = 1$, $\mu = 10/9$, and $\rho = 9/10$ for several values of k. The points that are plotted have been connected by lines. As expected, despite the initial condition, all cases appear to move toward the steady-state value of 9 with increasing n. The horizontal axis is only limited to $n = 120$ for display purposes and in fact, identical computations were carried out for $n > 300$ customers to verify convergence. However, as shown in the cases where $k = 6$ and $k = 10$, the convergence to steady-state is not always monotone. Additionally, in testing various traffic intensities, the rate of convergence to steady-state increases rapidly with decreasing traffic intensity for varying values of k.

APPL also has the ability to calculate the closed-form cumulative distribution function (CDF) for the nth customer's sojourn time permitting CDF comparisons for varying n as well as distribution percentiles for a given customer. The procedure call `CDF(T)` returns the exact CDF for customer 4 (from Example 1). Figure 6.2 displays the sojourn time CDF for varying n with fixed $k = 0$ and $\rho = 9/10$. The differences in CDFs across n correspond to the increasing mean attributed to the delays experienced by successive customers, e.g., customer 2 has delay time zero or exponential(μ) whereas the nth customer (for $n > 2$) faces a finite mixture of n potential delay distributions. The CDF associated with $n = \infty$ corresponds to the

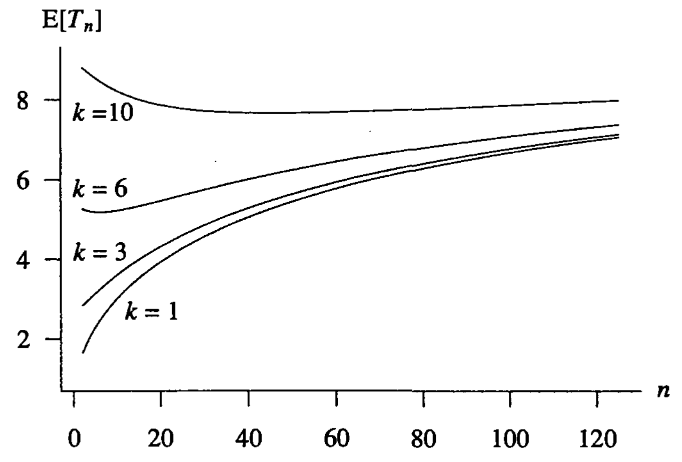


Figure 6.1: $M/M/1$ mean sojourn time for $\rho = 9/10$ given k at $t = 0$.

steady-state distribution of the sojourn time, which is exponentially distributed with a mean of 9 (Kleinrock, 1975).

Varying k for an $M/M/1$ queue also provides another basis for comparison of CDFs. Figure 6.3 fixes $n = 2$, $\rho = 9/10$, and plots the resulting CDFs across k. Kelton and Law (1985) make a similar comparison using convergence to steady state delay time. Using the CDF for multiple values of k allows direct comparison of sojourn time percentiles for customer n. As depicted, the sojourn time CDF for customer 2 is extremely sensitive to the initial condition k. As an illustration, the 80th percentiles for $k = 0, 3, 6$ are

$$F_2^{-1}(0.80) \approx \begin{cases} 1.935 & k = 0 \\ 4.432 & k = 3 \\ 7.510 & k = 6. \end{cases}$$

These percentiles are achieved using the APPL statements

```
X := ExponentialRV(1);
Y := ExponentialRV(10 / 9);
Z := Queue(X, Y, 2, k, 1);
IDF(Z, 0.8);
```

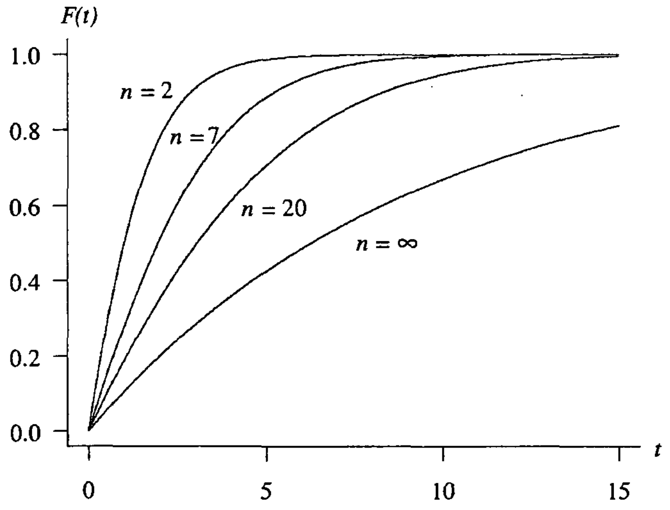


Figure 6.2: $M/M/1$ sojourn time CDFs for various n given $\rho = 9/10$ and $k = 0$.

when $k = 0, 3, 6$. The last statement, `IDF(Z, 0.8)`, numerically solves $F_Z(z) = 0.80$ on the interval $(0, \infty)$.

Given the complete specification of the sojourn time distribution, one can use APPL to calculate not only the mean but also the 2nd, 3rd, and 4th moments for customer n. This is especially valuable for steady-state analysis. It is common in simulation to verify attainment of steady-state behavior by examining the mean delay or mean sojourn time. Though some literature exists on estimating transient mean and variance, we are not aware of any literature addressing higher moments. Literature addressing the second moment seems mostly focused on variance estimation and not necessarily convergence. Therefore, even when the first moment might acceptably approximate the steady state value, there is reason for further analysis of higher moments. For example, Figure 6.4 displays the first four moments of the sojourn time for customer n in an $M/M/1$ queue, where $\lambda = 1$, $\mu = 2$, $\rho = 1/2$, with the initial condition $k = 0, 4, 8$. The steady-state values for the four measures of performance (the first four moments) are $1, 1, 2, 9$ The code used to calculate the values plotted in Figure 6.4 is

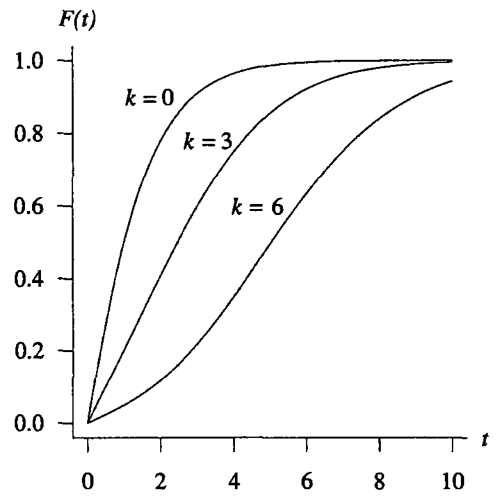


Figure 6.3: $M/M/1$ sojourn time CDFs for customer $n = 2$ for various k given $\rho = 9/10$.

```
X := ExponentialRV(1);
Y := ExponentialRV(2);
for i from 2 to 100 by 1 do
  T := Queue(X, Y, i, k, 1):
  print(i, evalf(Mean(T)), evalf(Variance(T)), evalf(Skewness(T)),
        evalf(Kurtosis(T))):
od:
```

for $k = 0, 4, 8$. The vertical dashed lines give the smallest customer number for which all three of the transient values are within 1% of the steady state value. The relatively low traffic intensity $\rho = 1/2$ was selected purposely to allow quick convergence and easy visual inspection. Even with this somewhat low traffic intensity, it is apparent that the higher moments converge more slowly than the lower moments. In other scenarios where $\rho > 1/2$, the higher moments exhibit an even slower convergence. Each vertical dashed line in Figure 6.4 was triggered by the $k = 8$ curve, suggesting that the moments are more sensitive to a heavily pre-loaded system. For the cases $k = 0, 4, 8$, the customer numbers for which the transient results were within 1% of the steady-state values are listed in Table 6.3. To verify the initial-condition effect on the convergence rate of the first four moments, k was increasingly incremented

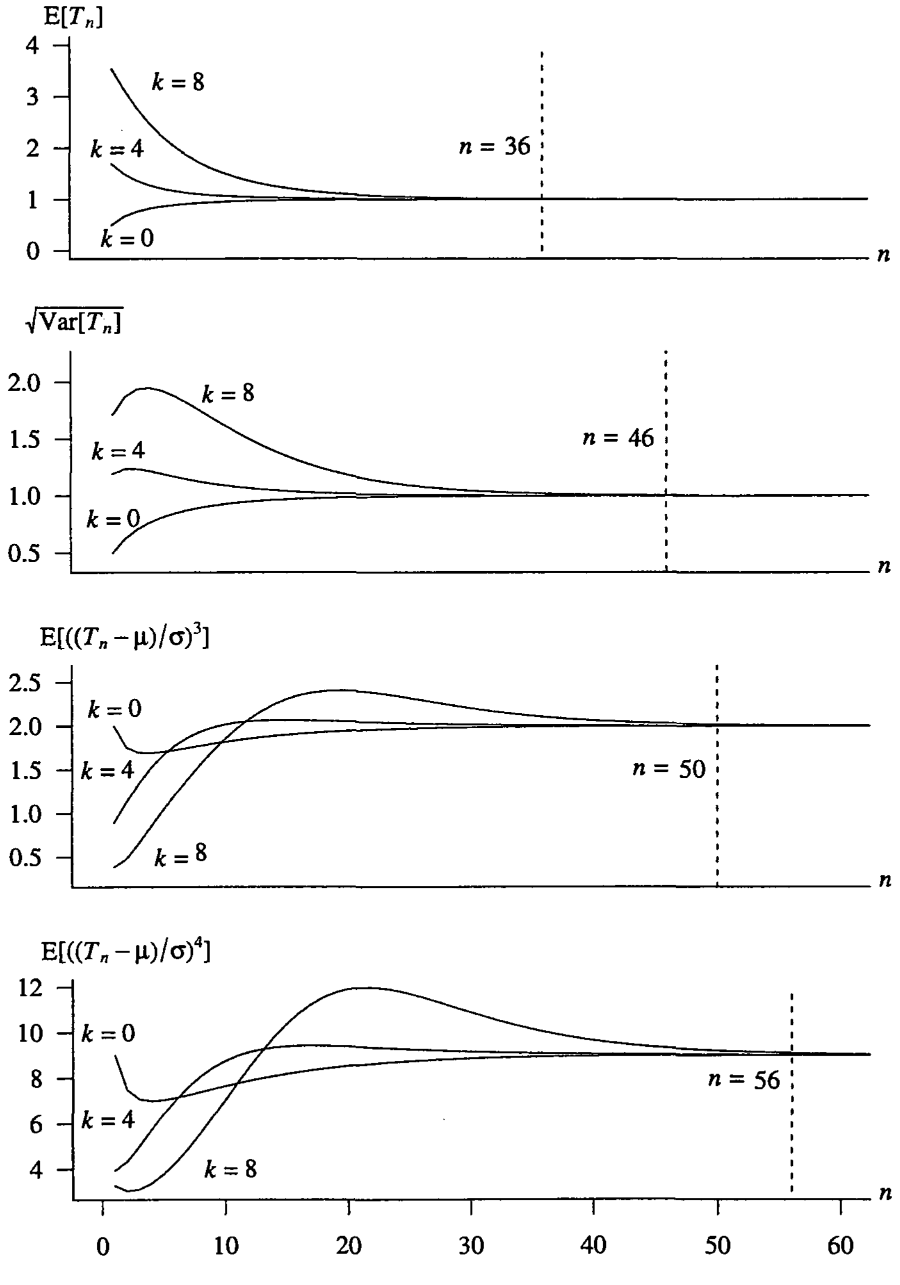


Figure 6.4: First four moments of the $M/M/1$ sojourn time for customers 2 through 100 for $\rho = 1/2$ and $k = 0, 4, 8$.

beyond eight and displayed a further slowing of convergence.

	$k=0$	$k=4$	$k=8$
$E[T]$	19	21	36
$\sqrt{\text{Var}[T]}$	27	29	46
$E\left[((T-\mu)/\sigma)^3\right]$	28	29	50
$E\left[((T-\mu)/\sigma)^4\right]$	34	35	56

Table 6.3: Smallest customer number where the sojourn time transient result is within 1% of steady state for an $M/M/1$ queue with $k = 0, 4, 8$ and $\rho = 1/2$.

6.5 Covariance and Correlation in the $M/M/1$ Queue

The dependence exhibited in sojourn times of successive customers is one reason for the difficulty in calculating interval estimators for queue measures of performance. In the simplest case, consider an empty and idle $M/M/1$ queue with interarrival and service rates λ and μ. Our desire is to calculate the covariance between the sojourn times of customers 1 and 2. Though the exact value of the covariance is available directly (and will be presented subsequently) we outline two approaches to simulate the result which are helpful in the presentation of the analytic result.

6.5.1 Discrete-Event Simulation

As previously discussed, customer 1 proceeds directly to service and two cases exist for customer 2. In the first case, customer 2 proceeds directly to service. In the second case, he delays until customer 1's departure. Both cases are shown in Figure 6.5. This subsection introduces two simulation approaches for generating the first two customer sojourn times.

The first approach is a standard discrete-event simulation model. Without loss of generality, assume that customer 1 arrives at time 0. In the next-event approach, a service time is generated for customer 1 according to the service distribution, exponential(μ), and an arrival time, a_2, for customer 2 is generated according to the time between arrivals distribution, exponential(λ). If the arrival occurs after customer 1's service completion, then customer 2 is also assigned an independent

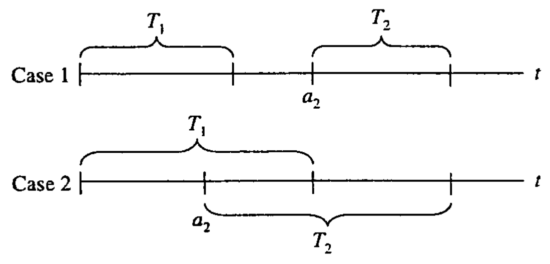


Figure 6.5: Discrete-event simulation model for cases 1 and 2.

exponential(μ) service time (case 1). In the second case in which customer 2's arrival time occurs before customer 1's completion of service ($a_2 < T_1$), customer 2 delays for $T_1 - a_2$ time units. We then add the exponential(μ) service time to the delay time to calculate T_2. We define the gap occurring in case two as, $Y = T_1 - a_2$. It can be shown analytically that $Y \sim$ exponential(μ) by computing the distribution of the difference $T_1 - A_2$, where A_2 is the random arrival time of the second customer and is distributed exponential(λ), and then truncating the result on the left at zero. (Alternately, it can be reasoned that $Y \sim$ exponential(μ) by the memoryless property for the exponential distribution since the remaining service time for customer 1 after customer 2's arrival has the same distribution as an unconditional service time.) Therefore, by using (6.1), in case 2 the sojourn time for customer 2 is distributed Erlang($\mu, 2$).

The second approach is a conditional discrete-event model, where the initial event, whose occurrence time is denoted as E_1 in Figure 6.6, is either a completion of service for customer 1 with probability $\mu/(\lambda+\mu)$ or the arrival of customer 2 with probability $\lambda/(\lambda + \mu)$. Since E_1 is the minimum of the arrival time of customer 2 and service time of customer 1, $E_1 \sim$ exponential($\lambda + \mu$). The R/S-Plus simulation code for each approach is listed in Appendix F. Using $n = 10,000,000$ replications, the two approaches are compared in Table 6.4. The simulation was run with three separate λ and μ pairs, capturing traffic intensities less than one, close to one, and greater than one. Though the two approaches displayed in Table 6.4 are fundamentally

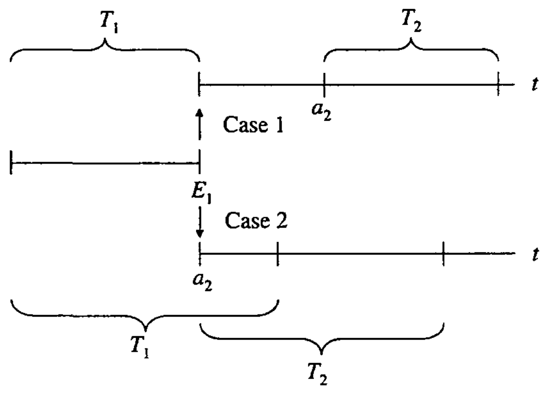


Figure 6.6: Conditional discrete-event simulation model for cases 1 and 2.

different, they are stochastically identical, so the resulting measures of performance are the same. Table 6.4 displays increasing correlation as traffic intensity increases. Scatterplots for $n = 1000$ (T_1, T_2) pairs are provided in Figure 6.7 for each (λ, μ) pair in Table 6.4. These correlation measures indicate the degree of dependence that occurs in the customers' sojourn times. As expected, in an unstable queue where $\rho > 1$, the correlation is highest.

A kernel density estimate of the joint distribution $f_{T_1,T_2}(t_1, t_2)$ from 10,000 pairs is plotted in Figure 6.8 for $\lambda = 1$ and $\mu = 1/2$. The estimate uses a normal kernel function with a smoothing parameter as prescribed in Bowman and Azzalini (1997). This three-dimensional image also indicates the relatively high correlation shown in Table 6.4 associated with this unstable traffic intensity.

6.5.2 Analytic Methods

One way to calculate the exact covariance between customers 1 and 2 requires the joint probability density function, $f_{T_1,T_2}(t_1, t_2)$. The method used here for computing the joint density uses Theorem 6.1 below.

$\lambda = 1$, $\mu = 2$	Approach 1	Approach 2
$E[T_1]$	0.500	0.500
$V[T_1]$	0.250	0.250
$E[T_2]$	0.666	0.666
$V[T_2]$	0.388	0.389
$E[Y]$	0.499	0.500
$V[Y]$	0.249	0.250
$E[T_2\|c_2]$	0.999	1.000
$V[T_2\|c_2]$	0.499	0.500
$\text{Cov}(T_1, T_2)$	0.138	0.139
$\text{Corr}(T_1, T_2)$	0.445	0.445
$\lambda = 1$, $\mu = 10/9$	Approach 1	Approach 2
$E[T_1]$	0.900	0.900
$V[T_1]$	0.809	0.810
$E[T_2]$	1.326	1.326
$V[T_2]$	1.395	1.395
$E[Y]$	0.900	0.900
$V[Y]$	0.809	0.809
$E[T_2\|c_2]$	1.800	1.799
$V[T_2\|c_2]$	1.619	1.619
$\text{Cov}(T_1, T_2)$	0.585	0.585
$\text{Corr}(T_1, T_2)$	0.551	0.550
$\lambda = 1$, $\mu = 1/2$	Approach 1	Approach 2
$E[T_1]$	1.999	2.000
$V[T_1]$	3.999	4.002
$E[T_2]$	3.333	3.334
$V[T_2]$	7.552	7.563
$E[Y]$	1.998	2.001
$V[Y]$	3.999	4.007
$E[T_2\|c_2]$	3.999	4.000
$V[T_2\|c_2]$	7.995	8.009
$\text{Cov}(T_1, T_2)$	3.549	3.561
$\text{Corr}(T_1, T_2)$	0.646	0.647

Table 6.4: Discrete-event simulation results using approaches 1 and 2.

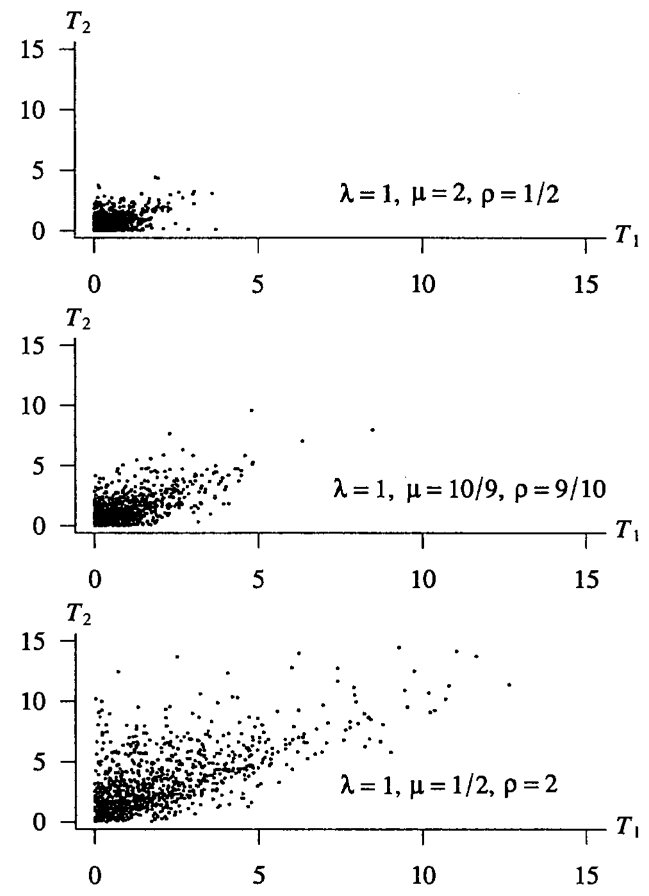


Figure 6.7: Scatterplots of the first two customer sojourn times in an $M/M/1$ queue.

Theorem 6.1 *Let $X_1 \sim$ exponential(λ_1), $X_2 \sim$ exponential(λ_2), and $X_3 \sim$ exponential(λ_3) be independent random variables. The joint probability density function of $(T_1, T_2) = (X_1 + X_2, X_1 + X_3)$ is*

$$f_{T_1,T_2}(t_1, t_2) = \begin{cases} \dfrac{\lambda_1\lambda_2\lambda_3\left(e^{\lambda_1 t_1} - e^{(\lambda_2+\lambda_3)t_1}\right)e^{-\lambda_1 t_1 - \lambda_2 t_1 - \lambda_3 t_2}}{\lambda_1 - \lambda_2 - \lambda_3} & 0 < t_1 < t_2 \\ \dfrac{\lambda_1\lambda_2\lambda_3\left(e^{\lambda_1 t_2} - e^{(\lambda_2+\lambda_3)t_2}\right)e^{-\lambda_2 t_1 - \lambda_1 t_2 - \lambda_3 t_2}}{\lambda_1 - \lambda_2 - \lambda_3} & 0 < t_2 < t_1. \end{cases}$$

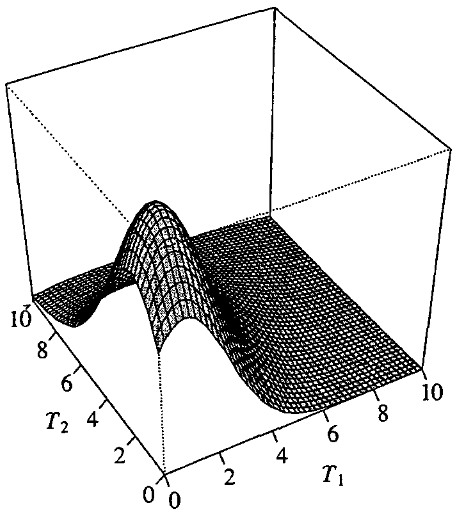


Figure 6.8: Kernel density estimate of $f_{T_1,T_2}(t_1,t_2)$ for $\lambda = 1$ and $\mu = 1/2$ from 10,000 simulated pairs.

Proof The joint CDF of T_1 and T_2 is

$$\begin{aligned}
F_{T_1,T_2}(t_1,t_2) &= \Pr(T_1 \le t_1, T_2 \le t_2) \\
&= \Pr(X_1 + X_2 \le t_1, X_1 + X_3 \le t_2) \\
&= \Pr(X_2 \le t_1 - X_1, X_3 \le t_2 - X_1) \\
&= \int_0^{\min\{t_1,t_2\}} \Pr(X_2 \le t_1 - x_1, X_3 \le t_2 - x_1 | X_1 = x_1) f_{X_1}(x_1) dx_1 \\
&= \int_0^{\min\{t_1,t_2\}} \Pr(X_2 \le t_1 - x_1 | X_1 = x_1) \Pr(X_3 \le t_2 - x_1 | X_1 = x_1) \cdot \\
&\qquad f_{X_1}(x_1) dx_1 \\
&= \int_0^{\min\{t_1,t_2\}} \left(1 - e^{-\lambda_2(t_1 - x_1)}\right)\left(1 - e^{-\lambda_3(t_2 - x_1)}\right) \lambda_1 e^{-\lambda_1 x_1} dx_1 \\
&= \begin{cases} \int_0^{t_1} \left(1 - e^{-\lambda_2(t_1 - x_1)}\right)\left(1 - e^{-\lambda_3(t_2 - x_1)}\right) \lambda_1 e^{-\lambda_1 x_1} dx_1 & 0 < t_1 < t_2 \\ \int_0^{t_2} \left(1 - e^{-\lambda_2(t_1 - x_1)}\right)\left(1 - e^{-\lambda_3(t_2 - x_1)}\right) \lambda_1 e^{-\lambda_1 x_1} dx_1 & 0 < t_2 < t_1. \end{cases}
\end{aligned}$$

After evaluating the integrals and differentiating, $f_{T_1,T_2}(t_1,t_2)$ is

$$f_{T_1,T_2}(t_1,t_2) = \begin{cases} \dfrac{\lambda_1\lambda_2\lambda_3\left(e^{\lambda_1 t_1} - e^{(\lambda_2+\lambda_3)t_1}\right)e^{-\lambda_1 t_1-\lambda_2 t_1-\lambda_3 t_2}}{\lambda_1-\lambda_2-\lambda_3} & 0 < t_1 < t_2 \\ \dfrac{\lambda_1\lambda_2\lambda_3\left(e^{\lambda_1 t_2} - e^{(\lambda_2+\lambda_3)t_2}\right)e^{-\lambda_2 t_1-\lambda_1 t_2-\lambda_3 t_2}}{\lambda_1-\lambda_2-\lambda_3} & 0 < t_2 < t_1. \end{cases} \quad \blacksquare$$

Theorem 6.1 provides the joint PDF of the first two sojourn times for case 2, which must be weighted appropriately by the probability that the arrival of customer 2 occurs prior to customer 1's completion of service, or $\lambda/(\lambda+\mu)$. Case 1 consists of independent sojourn times, so the joint density can be written as the product of the densities of the sojourn times T_1 and T_2 and weighted by $\mu/(\lambda+\mu)$. The resulting joint density is a mixture of the two possible cases displayed in Figure 6.6. We apply Theorem 1 to case 2 because of the dependence that occurs due to the overlap of the sojourn times. Figure 6.9 depicts the relationships between the sojourn times T_1, T_2 and the random variables X_1, X_2, and X_3 used in Theorem 1.

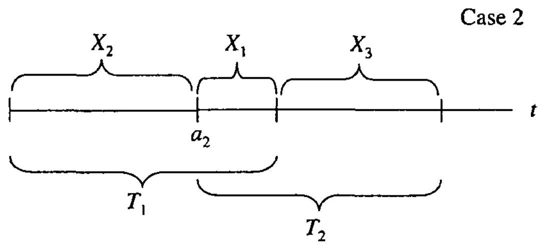


Figure 6.9: Case 2 for Theorem 1 with $X_1 \sim$ exponential(λ_1), $X_2 \sim$ exponential(λ_2), and $X_3 \sim$ exponential(λ_3).

Substituting $\lambda_1 = \mu$, $\lambda_2 = \lambda+\mu$, and $\lambda_3 = \mu$ into the mixture of cases 1 and 2 yields the joint PDF of T_1 and T_2 as

$$f_{T_1,T_2}(t_1,t_2) = \begin{cases} \dfrac{\mu^2\left(\lambda e^{-\mu t_2} + \mu e^{-\lambda t_1-\mu t_1-\mu t_2}\right)}{\lambda+\mu} & 0 < t_1 \le t_2 \\ \dfrac{\mu^2\left(\lambda e^{-\lambda t_1-\mu t_1+\lambda t_2} + \mu e^{-\lambda t_1-\mu t_1-\mu t_2}\right)}{\lambda+\mu} & 0 < t_2 < t_1. \end{cases} \tag{6.3}$$

Using this joint PDF, the covariance between the sojourn times of customers 1 and 2 is

$$\mathrm{Cov}(T_1, T_2) = \frac{\lambda(\lambda + 2\mu)}{(\lambda + \mu)^2 \mu^2}.$$

Substituting $\lambda = 1$ and $\mu = 2$, for example, produces

$$\mathrm{Cov}(T_1, T_2) = \frac{5}{36} \approx 0.13889,$$

which is consistent with the simulation results in Table 6.4. We now use the results of Theorem 6.1 in Example 4.

Example 4. Let T_1 and T_2 be the sojourn times for customers 1 and 2 respectively in an initially empty and idle $M/M/1$ queue with exponential(1) times between arrivals and exponential(2) service times. Find the distribution of the sample mean $\overline{T} = (T_1 + T_2)/2$ as well as $E[\overline{T}]$ and $V[\overline{T}]$.

Applying equation (6.3) with $\lambda = 1$ and $\mu = 2$, the joint PDF of T_1 and T_2 is

$$f_{T_1,T_2}(t_1, t_2) = \begin{cases} \frac{8}{3}e^{-3t_1-2t_2} + \frac{4}{3}e^{-2t_2} & 0 < t_1 \le t_2 \\ \frac{8}{3}e^{-3t_1-2t_2} + \frac{4}{3}e^{-3t_1+t_2} & 0 < t_2 < t_1. \end{cases}$$

Define the transformation

$$U = \overline{T} = (T_1 + T_2)/2 \qquad \text{and} \qquad V = (T_1 - T_2)/2$$

with inverse

$$T_1 = U + V \qquad \text{and} \qquad T_2 = U - V.$$

It can be shown that the functions U and V define a one-to-one transformation, thus, using the bivariate transformation technique described in

Hogg et al. (2005), the joint PDF of U and V is

$$f_{U,V}(u,v) = \begin{cases} f_{T_1,T_2}(u+v, u-v)|J| & -u \leq v < 0 \\ f_{T_1,T_2}(u+v, u-v)|J| & 0 < v < u, \end{cases}$$

where J is the Jacobian of the inverse transformation defined as

$$J = \begin{vmatrix} \dfrac{\partial t_1}{\partial u} & \dfrac{\partial t_1}{dv} \\ \dfrac{\partial t_2}{\partial u} & \dfrac{\partial t_2}{\partial v} \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & -1 \end{vmatrix} = -2.$$

Substituting $t_1 = u + v$, $t_2 = u - v$, $J = -2$ and integrating out the dummy transformation variable v, the resulting PDF of $U = \overline{T}$ is

$$f_U(u) = 4e^{-4u} + 2e^{-2u} - 6e^{-6u} \qquad u > 0.$$

The mean of U is

$$\begin{aligned} E[U] &= \int_0^\infty u \cdot f_U(u) du \\ &= \int_0^\infty u \cdot \left(4e^{-4u} + 2e^{-2u} - 6e^{-6u}\right) du \\ &= \frac{7}{12.} \end{aligned}$$

Likewise, the variance of U using $V[U] = E[U^2] - (E[U])^2$, where

$$\begin{aligned} E\left[U^2\right] &= \int_0^\infty u^2 \cdot f_U(u) du \\ &= \int_0^\infty u^2 \cdot \left(4e^{-4u} + 2e^{-2u} - 6e^{-6u}\right) du \\ &= \frac{41}{72}, \end{aligned}$$

results in

$$V[U] = \frac{41}{72} - \left[\frac{7}{12}\right]^2 = \frac{11}{48}.$$

Using the `Queue(X, Y, n, k, s)` procedure for customers 1 and 2, the mean sojourn times are $E[T_1] = 1/2$ and $E[T_2] = 2/3$ and the corresponding variances are $V[T_1] = 1/4$ and $V[T_2] = 7/18$. The covariance of sojourn times T_1 and T_2 was identified as $\text{Cov}(T_1, T_2) = 5/36$. Therefore, the mean sojourn time for customers 1 and 2 is

$$E\left[\frac{T_1 + T_2}{2}\right] = \frac{E[T_1] + E[T_2]}{2} = \frac{7}{12},$$

and the variance is

$$V\left[\frac{T_1 + T_2}{2}\right] = \frac{V[T_1] + V[T_2] + 2\text{Cov}(T_1, T_2)}{4} = \frac{11}{48},$$

further substantiating the distribution of $U = \overline{T}$ given above.

Proceeding in this manner, we now derive similar expressions for the first three customers arriving to an empty and idle $M/M/1$ queue. We could use first principles to derive the trivariate PDF $f_{T_1,T_2,T_3}(t_1, t_2, t_3)$; however, since covariance only occurs between two customers, it is easier to calculate each respective paired joint distribution for covariance calculations. A derivation of the trivariate distribution is provided in Appendix G; using the three variable distribution provides identical covariance results. However, calculating this trivariate joint distribution is tedious, and because the number of cases increases with the number of customers (as will be shown subsequently), the distribution complexity increases. When considering $n = 3$ customers, there are five possible ways customers can arrive and be serviced. In general, for n customers, the number of ways arrivals and departures can occur is given by the nth Catalan number, which is

$$C_n = \frac{(2n)!}{(n!)(n+1)!}.$$

Figure 6.10 shows the five possible arrangements for $n = 3$ customers along with the sojourn times T_1, T_2, and T_3 for each, with the arrival and completion times for the ith customer denoted by a_i and c_i respectively. The vertical arrows at event times represent service completions (pointing up) or arrivals (pointing down). This competing-event approach parallels the second simulation algorithm from Section 6.5.1. Using

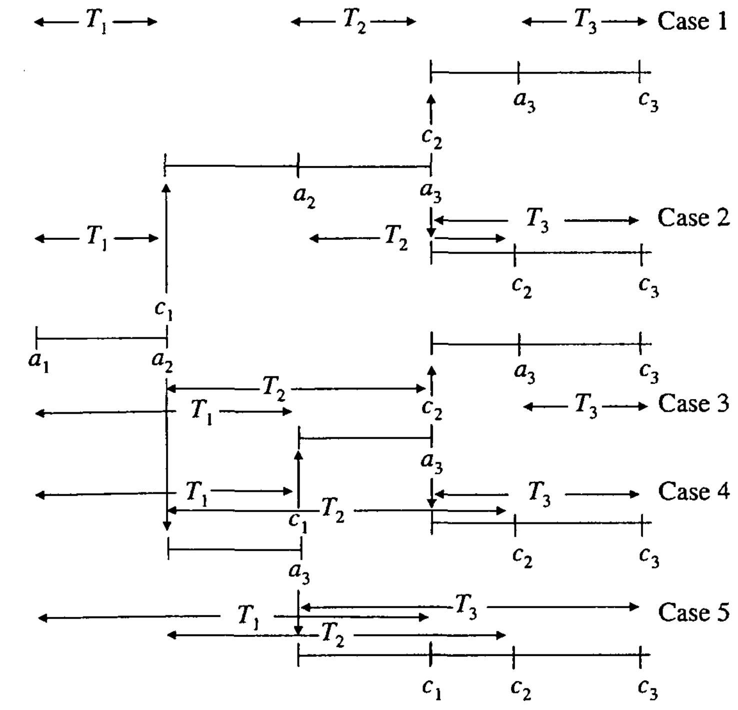


Figure 6.10: Five cases for $n = 3$ customers' sojourn times in an $M/M/1$ queue.

the same conditioning approach as in the proof of Theorem 1, the joint PDFs for each of the pairs (T_1, T_2), (T_1, T_3), and (T_2, T_3) in each of the five cases can be determined and then mixed to achieve the three associated joint PDFs. The mixture probabilities are calculated by multiplying the appropriate number of competing arrivals (with probability $\lambda/(\lambda+\mu)$) or service completions (with probability $\mu/(\lambda+\mu)$). For example, in case 1 shown in Figure 6.10, there are two instances with competing risks, both of which result in a service completion, thus the probability of this case is $\mu^2/(\lambda+\mu)^2$. Using these joint densities, the symmetric $n = 3$ variance–covariance matrix

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \sigma_{13} \\ \sigma_{12} & \sigma_2^2 & \sigma_{23} \\ \sigma_{13} & \sigma_{23} & \sigma_3^2 \end{bmatrix}$$

is

$$\Sigma = \begin{bmatrix} \dfrac{1}{\mu^2} & \dfrac{\lambda(2\mu+\lambda)}{(\lambda+\mu)^2\mu^2} & \dfrac{\lambda^2(\lambda^2+4\lambda\mu+5\mu^2)}{(\lambda+\mu)^4\mu^2} \\ \bullet & \dfrac{2\lambda^2+4\lambda\mu+\mu^2}{(\lambda+\mu)^2\mu^2} & \dfrac{\lambda(2\lambda^2+8\lambda^2\mu+11\lambda\mu^2+2\mu^3)}{(\lambda+\mu)^4\mu^2} \\ \bullet & \bullet & \dfrac{3\lambda^6+18\lambda^5\mu+45\lambda^4\mu^2+54\lambda^3\mu^3+30\lambda^2\mu^4+8\lambda\mu^5+\mu^6}{(\lambda+\mu)^6\mu^2} \end{bmatrix}.$$

Substituting $\lambda = 1$ and $\mu = 2$, for example, results in

$$\Sigma = \begin{bmatrix} \dfrac{1}{4} & \dfrac{5}{36} & \dfrac{29}{324} \\ \bullet & \dfrac{7}{18} & \dfrac{13}{54} \\ \bullet & \bullet & \dfrac{1451}{2916} \end{bmatrix} \approx \begin{bmatrix} 0.2500 & 0.1389 & 0.0895 \\ \bullet & 0.3889 & 0.2407 \\ \bullet & \bullet & 0.4976 \end{bmatrix}.$$

These results have been verified via Monte Carlo for the first $n = 3$ customers in Appendix H. The sojourn time variance increases with customer number down the diagonal of the matrix because of the nature of the queueing process, where the sojourn time distribution for each additional customer is dependent on all the previous customers. On the other hand, the off-diagonal covariance entries in each row decrease with customer separation, for example $\sigma_{13} < \sigma_{12}$.

6.6 Extending Covariance Calculations

Consider the $n = 3$ case where all three customers arrive prior to the first customer's completion of service (this is Case 5 in Figure 6.10). Using a 1 to represent an arrival and a -1 for a departure, this sequence of arrivals and departures can be represented by the vector

$$\begin{bmatrix} 1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix}.$$

Figure 6.11 depicts this case as a path from the bottom left node to the top right node of the figure. Moving right in the figure indicates an arrival and moving up indicates a service completion. Diagonal moves are not permitted. Each of the five possible sequences of arrivals and departures for $n = 3$, shown in Figure 6.10, can be depicted by a specific path from the bottom left node to the top right node. The paths are shown collectively in Appendix I.

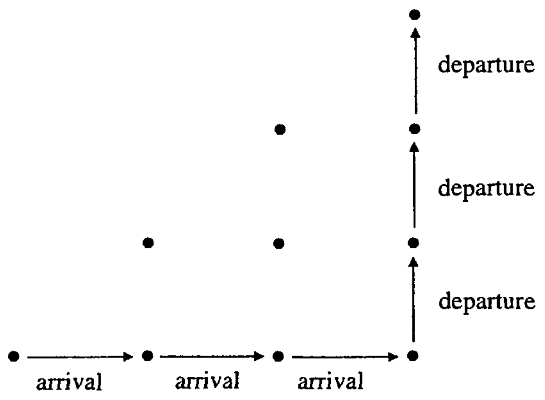


Figure 6.11: Path for case 5 of $n = 3$ customers arrival and departure pattern in an $M/M/1$ queue.

Ruskey and Williams (2008) present an elegant algorithm that generates all such paths of arrival and service completions for a given number of customers n. The algorithm is based on a simple iterative successor rule that uses prefix shifts (definition forthcoming) to exhaust the possible arrival and service completion scenarios. In Figure 6.11 these are the $6!/(3!4!) = 5$ paths that can be drawn from the bottom left node to the top right node without going above the diagonal line that connects these two nodes, and using only rightward and upward transitions. The algorithm is "loopless" in that it requires a constant amount of computation in transforming the current case to its successor. Define the case matrix C with dimension $(2n)!/((n!)(n+1)!)$ by $2n$ as the exhaustive list of possible arrival and service completion scenarios for n customers. To initiate the matrix the first row of C is

$$C_1 = \begin{bmatrix} 1 & -1 & 1 & 1 & -1 & -1 \end{bmatrix}.$$

The first row is always the ordered string created by an arrival, a service completion, $n-1$ arrivals, and $n-1$ service completions. The iterative successor rule described by Ruskey and Williams (2008) is: "Locate the leftmost $[-1, 1]$ and suppose its 1 is in position k. If the $(k+1)$-st prefix shift is valid (a possible arrival/service completion sequence), then it is the successor; if it is not valid then the k-th prefix shift is the successor." The $(k+1)$-st prefix shift for the sequence

$$B = \{B_1, B_2, \ldots, B_{k-1}, B_k, B_{k+1}, \ldots, B_{2n}\}$$

is

$$B = \{B_1, B_{k+1}, B_2, \ldots, B_{k-1}, B_k, B_{k+2}, \ldots, B_{2n}\}.$$

The length of the sequence is always $2n$ because the number of arrivals and departures is balanced at n each. An example of an invalid sequence is

$$\begin{bmatrix} 1 & -1 & -1 & 1 & 1 & -1 \end{bmatrix}$$

because the second service completion occurs prior to the second arrival. For $n = 3$, the case matrix C is

$$C = \begin{bmatrix} 1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix}.$$

(Note that the order of the five rows does not match the order of the cases in Figure 6.10.)

Figure 6.12 further categorizes each segment of the path based on whether there exists a competing risk (competing event) in which case the distribution of the time until the next event (either an arrival or a completion) is given by

$$\min\{\text{exponential}(\lambda),\ \text{exponential}(\mu)\} \sim \text{exponential}(\lambda + \mu),$$

where the time between arrivals is distributed as exponential(λ) and the service time distribution is exponential(μ).

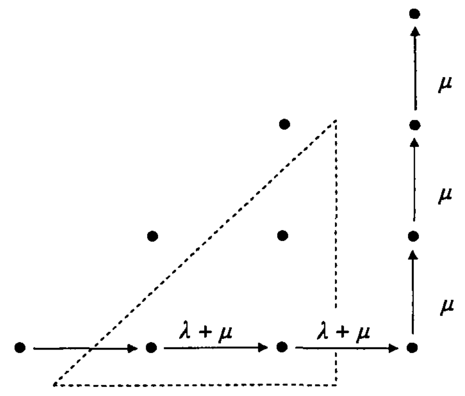


Figure 6.12: Path segment distributions for case 5 for $n = 3$ customers.

Competing risks can only occur along path segments originating inside the dashed triangle shown in Figure 6.12. These path segments are exponential(λ+μ) distributed and are correspondingly labeled $\lambda + \mu$. Once all customers have arrived, the only possible events are service completions; thus each vertical path segment along the rightmost edge of Figure 6.12 is distributed exponential(μ) and labeled μ. If the path of interest intersects the diagonal line that passes through the bottom left node and the top right node, the queueing system empties and the next event must be an arrival, which occurs in an exponential(λ) time into the future. While the system is empty, none of the customers' sojourn times are affected, therefore waiting for the next arrival does not impact customer sojourn time distribution. The interior triangle in the path diagram also provides a method to calculate the probability of all possible paths. For path segments originating inside the triangle, a move right occurs with probability $\lambda/(\lambda+\mu)$ and a move up occurs with probability $\mu/(\lambda+\mu)$. For the particular path shown in Figure 6.12 there are two segments originating inside the triangle, both of which are horizontal, representing two successive arrivals. Thus this case probability is

$$\frac{\lambda}{\lambda+\mu}\cdot\frac{\lambda}{\lambda+\mu}=\frac{\lambda^2}{(\lambda+\mu)^2}.$$

In order to capture the structure of the segment distributions for a given path, represented as a row of the case matrix C, another vector of length $2n-1$ is created where each entry corresponds to the sojourn time distribution for a particular segment. There are three possible entries in this vector:

1. exponential($\lambda+\mu$), which is indicated by a 1
2. exponential(μ), which is indicated by a 2
3. no distribution as a result of an emptied system, which is depicted as a 0.

The vector is of length $2n-1$ since the first customer's arrival time can be ignored as it does not affect sojourn time. For the particular path shown in Figure 6.12 the corresponding segment distribution vector is

$$\begin{bmatrix} 1 & 1 & 2 & 2 & 2 \end{bmatrix}.$$

Define the new matrix C' with dimension $(2n)!/((n!)(n+1)!)$ by $2n-1$ as the segment distribution matrix for each case in C. For $n=3$, the matrix C' is

$$C' = \begin{bmatrix} 1 & 0 & 1 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 \\ 1 & 0 & 1 & 0 & 2 \\ 1 & 1 & 1 & 0 & 2 \\ 1 & 1 & 2 & 2 & 2 \end{bmatrix}.$$

The two vectors, which are each the fifth row of the corresponding matrices

$$C_5 = \begin{bmatrix} 1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix} \quad \text{and} \quad C'_5 = \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \end{bmatrix}$$

contain the information necessary to calculate the contribution of Case 5 to the joint PDF for the sojourn times of any two customers. Using C_l, define the 2×2 matrix R_l with elements

$$R_l = \begin{bmatrix} r_{is} & r_{if} \\ r_{js} & r_{jf} \end{bmatrix}$$

where r_{is} and r_{if} are the start and finish indices for customer i in row l of the case matrix C. Define r_{js} and r_{jf} similarly for customer j. Using C_5 above, for customers $i=1$ and $j=3$,

$$R_5 = \begin{bmatrix} 1 & 4 \\ 3 & 6 \end{bmatrix}.$$

Customer 1's arrival is the first event to occur. Customer 1's departure is the fourth event to occur. Customer 3's arrival is the third event to occur. Customer 3's departure is the sixth event to occur.

The R_l matrix provides two critical pieces of information. First, for the given case l, if $r_{if} < r_{js}$ then the sojourn times for customers i and j are independent since customer i departs prior to customer j's arrival. Therefore, if $r_{if} < r_{js}$, the contribution of case l to the joint PDF is created by simply multiplying the sojourn time PDFs for customers i and j. Second, by computing $r_{if} - r_{is}$ and $r_{jf} - r_{js}$ and then indexing across C_l', the appropriate segment distributions can be combined to form the joint sojourn time PDF for customers i and j.

When $r_{if} > r_{js}$ the joint probability distribution is calculated by conditioning in a similar fashion to the proof of Theorem 1. However, it is first necessary to find the independent and overlapping segments for the customers of interest. For the arrival and service completion scenario described by C_5, Figure 6.13 shows sojourn times T_1 and T_3 for customers 1 and 3. The independent portion of customer 1's sojourn

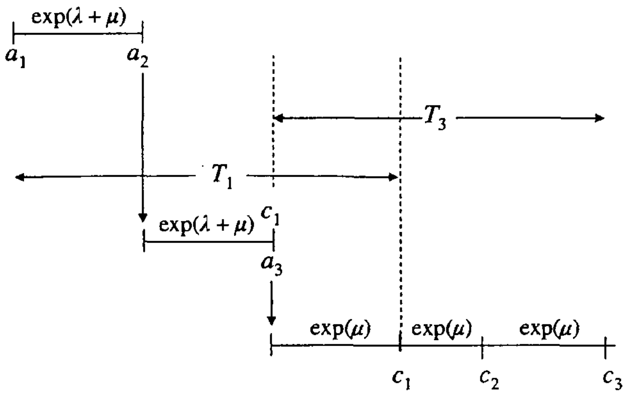


Figure 6.13: Sojourn time segments for customers 1 and 3 in case 5 of $n = 3$ customers.

time consists of the two exponential($\lambda + \mu$) segments. The independent portion of customer 3's sojourn time consists of the two exponential(μ) segments shown on the right side of Figure 13. The dependent (overlap) portion between customers 1 and 3 consists of the single exponential(μ) segment falling within the dashed vertical lines. Using C_5' and R_5, these segments can be determined without reference to Figure 6.13, as follows: Given $r_{1f} > r_{3s}$, that is customer 3 arrives prior to customer 1 completing service, the independent portions of customer 1's sojourn time distribution are found by (a) calculating $r_{3s} - r_{1s} = 3 - 1 = 2$ and then (b) collecting the elements in C_5' beginning at index $r_{1s} = 1$ and indexing $r_{3s} - r_{1s} - 1 = 1$ additional element of the vector. For $C_5' = \begin{bmatrix} 1 & 1 & 2 & 2 & 2 \end{bmatrix}$, the first two entries, c_{51}' and c_{52}' correspond to the two exponential($\lambda + \mu$) segments. Likewise, customer 3's independent sojourn time segments are found by (a) calculating $r_{3f} - r_{1f} = 6 - 4 = 2$ and then (b) collecting the elements in C_5' beginning at index $r_{1f} = 4$ and indexing $r_{3f} - r_{1f} - 1 = 1$ additional element of the vector. This amounts to the two exponential(μ) segments in elements four and five of C_5'. The dependent portion is identified by starting at the element $r_{3s} = 3$ and indexing $r_{if} - r_{3s} - 1 = 0$ additional elements, the third element of C_5', a single exponential(μ) segment.

In this case, calculating the joint PDF is straightforward since the independent portions for each customer are *iid* exponential random variables. Defining the independent cumulative distribution function portions for customers 1 and 3 as $X_1 \sim$ Erlang($\lambda + \mu, 2$) and $X_3 \sim$ Erlang($\mu, 2$) respectively, and the dependent (overlap) random variable as $W \sim$ exponential(μ), the contribution of Case 5 to the joint CDF of $(T_1, T_3) = (X_1 + W, X_3 + W)$, conditioning on the dependent distribution segment W, is

$$\begin{aligned}
F_{T_1,T_3}(t_1,t_3) &= P(T_1 \le t_1, T_3 \le t_3) \\
&= P(X_1 + W \le t_1, X_3 + W \le t_3) \\
&= P(X_1 \le t_1 - W, X_3 \le t_3 - W) \\
&= \int_0^{\min\{t_1,t_3\}} P(X_1 \le t_1 - w, X_3 \le t_3 - w | W = w) f_W(w) dw \\
&= \int_0^{\min\{t_1,t_3\}} P(X_1 \le t_1 - w | W = w) P(X_3 \le t_3 - w | W = w) f_W(w) dw \\
&= \int_0^{\min\{t_1,t_3\}} F_{X_1}(t_1 - w) F_{X_3}(t_3 - w) \mu e^{-\mu w} dw \\
&= \begin{cases} \int_0^{t_1} F_{X_1}(t_1 - w) F_{X_3}(t_3 - w) \mu e^{-\mu w} dw & 0 < t_1 < t_3 \\ \int_0^{t_3} F_{X_1}(t_1 - w) F_{X_3}(t_3 - w) \mu e^{-\mu w} dw & 0 < t_3 < t_1. \end{cases}
\end{aligned}$$

Since closed-form versions of $F_{X_1}(t_1 - w)$ and $F_{X_3}(t_3 - w)$ are available, Maple is capable of evaluating this expression, though for large n it can be time consuming.

When the independent distribution segments are not *iid* exponential random variables, the calculation is more problematic since we can no longer use (6.1) to easily express $F_{X_1}(t_1 - w)$ and $F_{X_3}(t_3 - w)$. Convolution is required, and though capable, Maple, and subsequently APPL, slow very quickly with increasing n. To overcome this shortfall, consider Theorem 6.2, which appears to be a faster approach than the two suggested in Hagwood (2009).

Theorem 6.2 *If $S_1 \sim$ Erlang(λ_1, m) and $S_2 \sim$ Erlang(λ_2, n) are independent random variables, then the PDF of $Y = S_1 + S_2$ is*

$$\begin{aligned} f_Y(y) &= \left[\frac{\lambda_1^m \lambda_2^n e^{-\lambda_2 y}}{(m-1)!(n-1)!} \sum_{x=0}^{n-1} \left\{(-1)^x \binom{n-1}{x} y^{n-1-x} e^{(\lambda_2-\lambda_1)s} \cdot \right.\right. \\ & \qquad \left.\left. \sum_{r=0}^{m-1+x} (-1)^r \frac{(m-1+x)! s^{m-1+x-r}}{(m-1+x-r)!(\lambda_2-\lambda_1)^{r+1}} \right\}\right]_{s=0}^{y} \qquad y > 0. \end{aligned}$$

Proof Since S_1 and S_2 are independent, the PDF of $Y = S_1 + S_2$ using convolution and the binomial theorem is

$$\begin{aligned} f_Y(y) &= \int_0^y f_{S_1}(s) f_{S_2}(y-s)\,ds \\ &= \int_0^y \frac{\lambda_1(\lambda_1 s)^{m-1} e^{-\lambda_1 s}}{(m-1)!} \frac{\lambda_2(\lambda_2(y-s))^{n-1} e^{-\lambda_2(y-s)}}{(n-1)!} ds \\ &= \frac{\lambda_1^m \lambda_2^n}{(m-1)!(n-1)!} \int_0^y s^{m-1} e^{-\lambda_1 s} (y-s)^{n-1} e^{-\lambda_2(y-s)} ds \\ &= \frac{\lambda_1^m \lambda_2^n e^{-\lambda_2 y}}{(m-1)!(n-1)!} \int_0^y s^{m-1} (y-s)^{n-1} e^{s(\lambda_2-\lambda_1)} ds \\ &= \frac{\lambda_1^m \lambda_2^n e^{-\lambda_2 y}}{(m-1)!(n-1)!} \int_0^y s^{m-1} \left(\sum_{x=0}^{n-1} \binom{n-1}{x} y^{n-1-x} (-s)^x\right) e^{s(\lambda_2-\lambda_1)} ds \\ &= \frac{\lambda_1^m \lambda_2^n e^{-\lambda_2 y}}{(m-1)!(n-1)!} \sum_{x=0}^{n-1} \left\{(-1)^x \binom{n-1}{x} y^{n-1-x} \int_0^y s^{m-1+x} e^{s(\lambda_2-\lambda_1)} ds\right\} \\ &= \left[\frac{\lambda_1^m \lambda_2^n e^{-\lambda_2 y}}{(m-1)!(n-1)!} \sum_{x=0}^{n-1} \left\{(-1)^x \binom{n-1}{x} y^{n-1-x} e^{(\lambda_2-\lambda_1)s} \cdot \right.\right. \\ & \qquad \left.\left. \sum_{r=0}^{m-1+x} (-1)^r \frac{(m-1+x)! s^{m-1+x-r}}{(m-1+x-r)!(\lambda_2-\lambda_1)^{r+1}} \right\}\right]_{s=0}^{y} \qquad y > 0. \qquad \blacksquare \end{aligned}$$

The APPL procedure `Cov(a, b, n)` applies Theorem 6.2 to calculate the covariance between customers a and b ($a < b$) in a system of n customers. For computational considerations (i.e., evaluating the fewest cases necessary for a given n), setting the number of customers $n = b$ provides the fastest result. Additionally, calling `Cov(a, b, n)` where $n > b$ produces a result identical to $n = b$ because customers arriving after customer b do not affect the covariance of previous customers. The commented procedure is available in Appendix J.

Rewriting the integral as a sum via Theorem 6.2 avoids the calls to `Convolution(X, Y)` in APPL as well as integrating for each case and piece, and the speed-up was significant. One can always use this approach, even when the independent part of a particular customer's sojourn time contains many independent distribution segments. The times for these segments can only be exponential($\lambda + \mu$) distributed or exponential(μ) distributed, implying their sum can always be written as the sum of two independent Erlang random variables. The symmetric variance–covariance matrix for $n = 10$ customers with parameters $\lambda = 1$, $\mu = 2$, and $\rho = 1/2$ is showcased in Table 6.5 providing the exact values. CPU time is a factor in these computations. Each element in the tenth column of the variance–covariance matrix is calculated from a joint PDF which is a mixture of $C_{10} = 20!/(10!11!) = 16,796$ component distributions, each corresponding to a unique ordering of arrivals and departures.

$\frac{1}{4}$	$\frac{5}{36}$	$\frac{29}{324}$	$\frac{181}{2916}$	$\frac{1181}{26244}$	$\frac{2647}{78732}$	$\frac{18191}{708588}$	$\frac{127111}{6377292}$	$\frac{2699837}{172186884}$	$\frac{19319845}{1549681956}$
•	$\frac{7}{18}$	$\frac{13}{54}$	$\frac{239}{1458}$	$\frac{1543}{13122}$	$\frac{10303}{118098}$	$\frac{23485}{354294}$	$\frac{163493}{3188646}$	$\frac{3462503}{86093442}$	$\frac{24719519}{774840978}$
•	•	$\frac{1451}{2916}$	$\frac{8531}{26244}$	$\frac{53995}{236196}$	$\frac{356291}{2125764}$	$\frac{805705}{6377292}$	$\frac{5576849}{57395628}$	$\frac{39197977}{516560652}$	$\frac{836647331}{13947137604}$
•	•	•	$\frac{34514}{59049}$	$\frac{209794}{531441}$	$\frac{1357010}{4782969}$	$\frac{3031606}{14348907}$	$\frac{20810726}{129140163}$	$\frac{145390102}{1162261467}$	$\frac{3088887890}{31381059609}$
•	•	•	•	$\frac{12525605}{19131876}$	$\frac{77889229}{172186884}$	$\frac{170586983}{516560652}$	$\frac{1156711327}{4649045868}$	$\frac{8013045911}{41841412812}$	$\frac{169183999981}{1129718145924}$
•	•	•	•	•	$\frac{551583889}{774840978}$	$\frac{1162296371}{2324522934}$	$\frac{7727099083}{20920706406}$	$\frac{52871149859}{188286357654}$	$\frac{1106749378225}{5083731656658}$
•	•	•	•	•	•	$\frac{10582107143}{13947137604}$	$\frac{67728246079}{125524238436}$	$\frac{454382575415}{1129718145924}$	$\frac{9394007745229}{30502389939948}$
•	•	•	•	•	•	•	$\frac{225196533287}{282429536481}$	$\frac{1455144635743}{2541865828329}$	$\frac{29498588275973}{68630377364883}$
•	•	•	•	•	•	•	•	$\frac{75890492486993}{91507169819844}$	$\frac{1482244865480580}{2470693585135780}$
•	•	•	•	•	•	•	•	•	$\frac{28549065408995300}{33354363399333100}$

Table 6.5: Sojourn time variance–covariance matrix for the first $n = 10$ customers in an $M/M/1$ queue with $\lambda = 1$, $\mu = 2$.

Because these values are difficult to compare in fractional form, the same matrix is provided again, with matrix elements rounded to four decimal places.

$$\begin{bmatrix}
0.2500 & 0.1389 & 0.0895 & 0.0621 & 0.0450 & 0.0336 & 0.0257 & 0.0199 & 0.0157 & 0.0125 \\
\bullet & 0.3889 & 0.2407 & 0.1639 & 0.1176 & 0.0872 & 0.0663 & 0.0513 & 0.0402 & 0.0319 \\
\bullet & \bullet & 0.4976 & 0.3251 & 0.2286 & 0.1676 & 0.1263 & 0.0972 & 0.0759 & 0.0600 \\
\bullet & \bullet & \bullet & 0.5845 & 0.3948 & 0.2837 & 0.2113 & 0.1611 & 0.1251 & 0.0984 \\
\bullet & \bullet & \bullet & \bullet & 0.6547 & 0.4524 & 0.3302 & 0.2488 & 0.1915 & 0.1498 \\
\bullet & \bullet & \bullet & \bullet & \bullet & 0.7119 & 0.5000 & 0.3694 & 0.2808 & 0.2177 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 0.7587 & 0.5396 & 0.4022 & 0.3080 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 0.7974 & 0.5725 & 0.4298 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 0.8293 & 0.5999 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 0.8559
\end{bmatrix}$$

As the traffic intensity increases, so do the values in the variance–covariance matrix. To illustrate, the same matrix is provided for the increased traffic intensity parameters $\lambda = 1$, $\mu = 10/9$, and $\rho = 9/10$. The increasing sojourn-time variance along the diagonal is expected with the increasing traffic intensity. In addition, the rate that covariance between customers decreases as customer separation increases is less pronounced.

$$\begin{bmatrix}
0.8100 & 0.5856 & 0.4737 & 0.4040 & 0.3553 & 0.3189 & 0.2904 & 0.2673 & 0.2481 & 0.2318 \\
\bullet & 1.3956 & 1.1097 & 0.9393 & 0.8226 & 0.7363 & 0.6692 & 0.6150 & 0.5702 & 0.5323 \\
\bullet & \bullet & 1.9561 & 1.6298 & 1.4167 & 1.2626 & 1.1441 & 1.0494 & 0.9714 & 0.9057 \\
\bullet & \bullet & \bullet & 2.5021 & 2.1458 & 1.8995 & 1.7142 & 1.5679 & 1.4484 & 1.3485 \\
\bullet & \bullet & \bullet & \bullet & 3.0364 & 2.6565 & 2.3831 & 2.1715 & 2.0009 & 1.8593 \\
\bullet & \bullet & \bullet & \bullet & \bullet & 3.5605 & 3.1614 & 2.8652 & 2.6310 & 2.4389 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 4.0754 & 3.6600 & 3.3444 & 3.0904 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 4.5818 & 4.1524 & 3.8199 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 5.0803 & 4.6386 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 5.5713
\end{bmatrix}$$

Using this variance–covariance matrix for traffic intensity $\rho = 9/10$, consider the following example.

Example 5. Let T_i, $i = 1, 2, \ldots, 10$, be the sojourn times for the first $n = 10$ customers in an $M/M/1$ queue with arrival rate $\lambda = 1$ and service rate $\mu = 10/9$ that is initially empty and idle. Find the variance of the average sojourn time for the ten customers.

Define the average sojourn time as

$$\overline{T} = \frac{1}{10} \sum_{i=1}^{10} T_i.$$

Since the sojourn times are not independent random variables, the variance of the average sojourn time is

$$\begin{aligned}\mathrm{Var}(\overline{T}) &= \mathrm{Var}\left(\frac{1}{10}\sum_{i=1}^{10} T_i\right) \\ &= \frac{1}{100}\mathrm{Var}\left(\sum_{i=1}^{10} T_i\right) \\ &= \frac{1}{100}\left[\sum_{i=1}^{10}\mathrm{Var}\left(T_i\right) + 2\sum\sum_{i<j}\mathrm{Cov}\left(T_i, T_j\right)\right].\end{aligned}$$

The result is the sum of all elements in the variance–covariance matrix multiplied by the constant 1/100. The sum of the variance–covariance matrix rounded to four significant digits is 177.6642; therefore the variance of $\overline{T}$ is

$$V\left(\overline{T}\right) \approx 1.7766.$$

To verify the calculation a Monte Carlo simulation (listed in Appendix K) was conducted five times, each using one million replications. The resulting 95% confidence interval for the variance of $\overline{T}$ was $\overline{T} \in (1.773, 1.781)$, which agrees with the analytic result.

Traditional steady-state queueing theory and analysis lacks the insight provided in these transient variance–covariance matrices. For businesses where the number of customers in a day is so small that true steady state is never achieved, routine queueing measures of performance are not representative of reality. Additionally, consider a system where the traffic intensity exceeds one. For a such a system, an analyst might be interested in customer covariance. Increasing the traffic intensity so that $\rho > 1$ does not preclude covariance calculations using this method, and therefore allows transient analysis of such systems. A variance–covariance matrix for $\rho = 3/2$, is presented below. Given this traffic intensity, the system is unstable and the expected sojourn times for successive customers increase without bound. Along the main diagonal the customer variance is clearly increasing, and the covariance decreases as the separation occurs between customers. This decrease is monotonic, and though not studied in detail here, it appears that the rate of covariance decrease might be of interest for an unstable traffic intensity.

$$\begin{bmatrix}
2.2500 & 1.8900 & 1.7172 & 1.6135 & 1.5438 & 1.4937 & 1.4558 & 1.4263 & 1.4027 & 1.3835 \\
\bullet & 4.1400 & 3.7368 & 3.5018 & 3.3459 & 3.2344 & 3.1507 & 3.0856 & 3.0337 & 2.9913 \\
\bullet & \bullet & 6.0957 & 5.6825 & 5.4166 & 5.2292 & 5.0896 & 4.9817 & 4.8958 & 4.8261 \\
\bullet & \bullet & \bullet & 8.1312 & 7.7208 & 7.4397 & 7.2332 & 7.0747 & 6.9493 & 6.8479 \\
\bullet & \bullet & \bullet & \bullet & 10.2424 & 9.8410 & 9.5538 & 9.3361 & 9.1652 & 9.0276 \\
\bullet & \bullet & \bullet & \bullet & \bullet & 12.4235 & 12.0342 & 11.7463 & 11.5230 & 11.3444 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 14.6687 & 14.2931 & 14.0081 & 13.7828 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 16.9727 & 16.6115 & 16.3319 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 19.3310 & 18.9846 \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & 21.7397
\end{bmatrix}$$

6.7 Sojourn Time Covariance with k Customers Initially Present

When k customers are present in the $M/M/1$ queue at time zero, the approach used to compute sojourn-time covariance between customers becomes more difficult. When the two customers of interest possess indices larger than k (i.e., T_i where $i > k$), then the approach is similar to that derived in Section 6.6. However, there are two other possibilities. The first possibility is that the first customer has an index of k or less, and the second customer has an index larger than k. In this instance, the only difference in deriving the joint CDF is that the lower indexed customer begins his sojourn time at time zero. In the second possibility, both customers have an index of k or below. If these indices are i and j, where $i < j \leq k$, the time intervals for sojourn times T_i and T_j begin at zero. It is obvious that $T_i \leq T_j$, since the completion time for customer i must occur prior to the completion time for customer j. For each of the possibilities above the covariance derivation that follows will mirror the empty and idle covariance derivation in Section 6.6.

To illustrate the calculations, consider an $M/M/1$ queue with $k = 2$ customers initially present at time zero and a single additional customer, $n = 1$. The transition diagram where the first event (not including the k customers initially present at time zero) is an arrival, which is analogous to Figures 6.11 and 6.12, is given in Figure 6.14. The total number of customers passing through the system is $n + k = 3$. Using 1

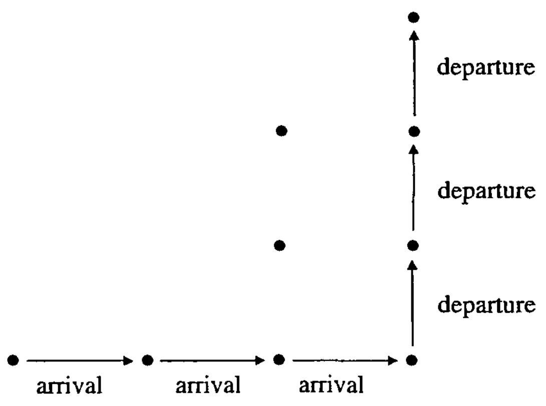


Figure 6.14: Transition diagram for $n + k = 1 + 2 = 3$ customers when the first event is an arrival.

to denote an arrival and -1 to denote a departure, each arrival/departure ordering instance for $n + k = 3$ customers must contain exactly three -1's (completions of service) and a single 1 (arrival). The algorithm presented by Ruskey and Williams (2008) does not facilitate listing all orderings for an unbalanced system, where the number of departures is greater than the number of arrivals (as opposed to an empty and idle queue at time zero). However, we can produce all possible arrival–departure sequences with a simple manipulation of the algorithm, as well as count the number of possible sequences. A derivation and proof of a formula for counting the number of possible sequences is provided in Appendix L. The general counting result, denoted by $C(n|k)$, follows, where n represents the number of customers passing through the system that arrive after time zero and k is the number of customers present at time zero:

$$C(n|k) = \sum_{j=0}^{\lfloor k/2 \rfloor} (-1)^j \binom{k-j}{j} C_{n+k-j}$$

for $k = 0, 1, 2, \ldots$ and $n = 1, 2, \ldots$, where $\lfloor \cdot \rfloor$ denotes the greatest integer function. The case matrix C is found by applying the Ruskey and Williams (2008) algorithm for $n+k$ customers, then deleting the instances where the first k events do not correspond to arrivals. As seen previously, the case matrix for $n + k = 1 + 2 = 3$ customers is

$$C = \begin{bmatrix} 1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix}.$$

Rows 2, 4, and 5 correspond to the first $k = 2$ events being arrivals. Rows 1 and 3 must be deleted from the case matrix, since for each row, a completion of service occurs prior to the first two arrivals. Deleting these rows results in the case matrix

$$C = \begin{bmatrix} 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix}$$

with the remaining rows representing all possible arrival–departure sequences. We can further simplify the case matrix by deleting the first k columns, resulting in

$$C = \begin{bmatrix} -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ 1 & -1 & -1 & -1 \end{bmatrix}.$$

The rows of the case matrix correspond to the three cases shown in Figure 6.15.

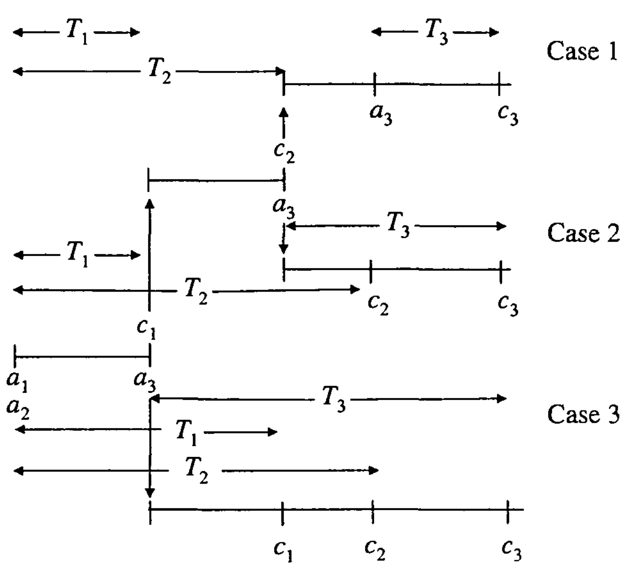


Figure 6.15: Three cases for $k = 2$ initial customers and a single $n = 1$ additional customer in an $M/M/1$ queue.

The algorithm for computing the joint PDF, and subsequently the covariance, of the sojourn times of any two customers does not differ significantly from the algorithm presented in Section 6.6. However, for the sojourn times T_1 and T_2 in Figure 6.15, a new theorem is introduced.

Theorem 6.3 *Let $X \sim$ exponential(λ_1) and $Y \sim$ exponential(λ_2) be independent random variables. The joint PDF of $(T_1, T_2) = (X, X + Y)$ is*

$$f_{T_1,T_2}(t_1, t_2) = \lambda_1 \lambda_2 e^{-\lambda_2 t_1 - \lambda_1 t_2 + \lambda_1 t_1} \qquad 0 < t_1 < t_2.$$

Proof The joint CDF of T_1 and T_2 is

$$\begin{aligned}
F_{T_1,T_2}(t_1,t_2) &= \Pr(T_1 \le t_1, T_2 \le t_2) \\
&= \Pr(X \le t_1, X+Y \le t_2) \\
&= \Pr(X \le t_1, Y \le t_2 - X) \\
&= \int_0^{t_1}\int_0^{t_2-x} f_X(x)\cdot f_Y(y)\,dy\,dx \\
&= \int_0^{t_1}\int_0^{t_2-x} \left(\lambda_1 e^{-\lambda_1 x}\right)\cdot\left(\lambda_2 e^{-\lambda_2 y}\right)dy\,dx \\
&= \frac{\lambda_1 - \lambda_2 + \lambda_2 e^{-\lambda_1 t_2} + \lambda_2 e^{-\lambda_2 t_1} - \lambda_1 e^{-\lambda_2 t_1} - \lambda_2 e^{-\lambda_2 t_1 - \lambda_1 t_2 + \lambda_1 t_1}}{\lambda_1 - \lambda_2},
\end{aligned}$$

for $0 < t_1 < t_2$. Taking partial derivatives, $f_{T_1,T_2}(t_1,t_2)$ is

$$f_{T_1,T_2}(t_1,t_2) = \lambda_1\lambda_2 e^{-\lambda_2 t_1 - \lambda_1 t_2 + \lambda_1 t_1} \qquad 0 < t_1 < t_2. \qquad \blacksquare$$

Theorem 6.3 provides the joint PDF for the sojourn times T_1 and T_2 of the first two customers initially present at time zero. It may be more complicated to calculate the joint PDFs for the sojourn times of other pairs of customers who were initially present at time zero. This is due to the fact that if $(i,j) \neq (1,2)$ and $i < j \le k$, where k is the number of customers present at time zero, the time intervals of duration X and Y during which customers i and j, respectively, are served may each be composed of multiple independent exponentially distributed time segments. Each of these multiple segments is limited to only one of two possibilities, an exponential($\lambda + \mu$) segment or an exponential(μ) segment. In this more complicated situation we let $(T_i, T_j) = (X, X+Y)$ as in Theorem 6.3 and apply Theorem 6.2 to quickly find the PDF's of X and Y (using the procedure `conv(m, n)`), then let Maple handle the sojourn time joint PDF calculation. When the second customer of interest has an index $\ge k$, the sojourn time joint PDF follows an application of Theorem 6.1 as described in Section 6.6 when cases exist with dependence.

Using the final case matrix C above, the associated segment distribution matrix

C' is

$$C' = \begin{bmatrix} 1 & 1 & 0 & 2 \\ 1 & 1 & 2 & 2 \\ 1 & 2 & 2 & 2 \end{bmatrix},$$

where the possible elements are the same as defined in Section 6.6. The probability vector associated with the case matrix C is

$$\begin{bmatrix} \frac{4}{9} & \frac{2}{9} & \frac{1}{3} \end{bmatrix}.$$

Using the case matrix C and the segment distribution matrix C', the joint PDFs for each case are created by selecting the appropriate segments for a given pair of customers, where the segments are identified by the R_l matrix discussed in Section 6.6. Once the joint PDF's are created for each case, they are mixed with the probability vector to determine the sojourn time joint PDF for covariance calculations. These calculations are coded in Maple as the procedure `kCov(X, Y, a, b, n, k)`. The first two arguments X and Y are the distribution of time between arrivals, exponential(λ), and the service time distribution, exponential(μ), respectively. They are entered in the APPL list-of-lists format. The arguments a and b are the customers of interest for the covariance calculation, where $a < b$. The argument n is the number of customers processing through the system not including those present at time zero, which is indicated by the last argument, k. Therefore, the total number of customers processing through the system is $n + k$, and a covariance calculation between any two of these customers can be achieved with the appropriate function call. For example, the function call `kCov(ExponentialRV(1), ExponentialRV(2), 1, 2, 6, 4)` calculates the covariance between customers 1 and 2 in an $M/M/1$ queue with an arrival rate $\lambda = 1$, with service time rate $\mu = 2$, with $k = 4$ customers present at time zero, and an additional $n = 6$ customers process through the system. The complete variance–covariance matrix using these paramters is

$$\begin{bmatrix}
\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{211}{972} & \frac{1579}{8748} & \frac{11651}{78732} & \frac{28553}{236196} & \frac{630131}{6377292} & \frac{4646155}{57395628} \\
\bullet & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{211}{486} & \frac{1579}{4374} & \frac{11651}{39366} & \frac{28553}{118098} & \frac{630131}{3188646} & \frac{4646155}{28697814} \\
\bullet & \bullet & \frac{3}{4} & \frac{3}{4} & \frac{211}{324} & \frac{1579}{2916} & \frac{11651}{26244} & \frac{28553}{78732} & \frac{630131}{2125764} & \frac{4646155}{19131876} \\
\bullet & \bullet & \bullet & 1 & \frac{211}{243} & \frac{1579}{2187} & \frac{11651}{19683} & \frac{28553}{59049} & \frac{630131}{1594323} & \frac{4646155}{14348907} \\
\bullet & \bullet & \bullet & \bullet & \frac{37289}{26244} & \frac{271153}{236196} & \frac{1966777}{2125764} & \frac{1588153}{2125764} & \frac{34755203}{57395628} & \frac{763875281}{1549681956} \\
\bullet & \bullet & \bullet & \bullet & \bullet & \frac{1629655}{1062882} & \frac{11663887}{9565938} & \frac{9353743}{9565938} & \frac{203800469}{258280326} & \frac{4465399991}{6973568802} \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \frac{263490131}{172186884} & \frac{208262483}{172186884} & \frac{4506205633}{4649045868} & \frac{98323535707}{125524238436} \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \frac{63939878}{43046721} & \frac{1359189250}{1162261467} & \frac{29402061622}{31381059609} \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \frac{179260456277}{125524238436} & \frac{379721786263}{3389154437772} \\
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \frac{62708955663745}{45753584909922}
\end{bmatrix}.$$

Unlike the previous variance–covariance matrices, some row elements, in particular those elements associated with customers that are initially present, do not decrease monotonically. To explain these entries, consider Theorem 6.4.

Theorem 6.4 *If $X_1, X_2, \ldots, X_n$ are iid exponential(μ) random variables and*

$$T_s = \sum_{r=1}^{s} X_r \qquad s = 1, 2, \ldots, n,$$

then $\text{Var}(T_i) = \text{Cov}(T_i, T_l), 0 < i < l \leq n$.

Proof Note that $E[T_k] = k/\mu$ for $k = 1, 2, \ldots, n$ and that T_i and X_r are independent for $1 \le i < r \le n$.

$$\begin{aligned}
\mathrm{Cov}(T_i, T_l) &= E\left[\left(T_i - \frac{i}{\mu}\right)\left(T_l - \frac{l}{\mu}\right)\right] \\
&= E\left[\left(T_i - \frac{i}{\mu}\right)\left\{\left(T_i - \frac{i}{\mu}\right) + \sum_{r=i+1}^{l}\left(X_r - \frac{1}{\mu}\right)\right\}\right] \\
&= E\left[\left(T_i - \frac{i}{\mu}\right)^2\right] + E\left\{\sum_{r=i+1}^{l}\left[\left(T_i - \frac{i}{\mu}\right)\left(X_r - \frac{1}{\mu}\right)\right]\right\} \\
&= \mathrm{Var}(T_i) + \sum_{r=i+1}^{l} E\left(T_i - \frac{i}{\mu}\right) E\left(X_r - \frac{1}{\mu}\right) \\
&= \mathrm{Var}(T_i). \qquad \blacksquare
\end{aligned}$$

We can apply Theorem 6.4 to those customer pairs where both indices $i, j \le k$. Therefore, the entries in the variance–covariance matrix for customer pairs $(1,2)$, $(1,3)$, and $(1,4)$ are

$$\mathrm{Var}(T_1) = \mathrm{Cov}(T_1, T_2) = \mathrm{Cov}(T_1, T_3) = \mathrm{Cov}(T_1, T_4) = \frac{1}{4}.$$

Likewise, for the customer pairs $(2,3)$ and $(2,4)$

$$\mathrm{Var}(T_2) = \mathrm{Cov}(T_2, T_3) = \mathrm{Cov}(T_2, T_4) = \frac{1}{2}.$$

Furthermore, it can be shown in general that

$$\mathrm{Var}(T_i) = \mathrm{Cov}(T_i, T_j) = \frac{i}{\mu^2}$$

for $i < j \le k$, where k customers are present at time zero. For example, consider a single-server box office with exponential(μ) service times which will be offering tickets to a popular concert the next day. If 1000 patrons, each buying one ticket, camp out the night before determined to get the best seats for the concert, these $k = 1000$ customers are present at time zero and therefore we can pre-determine the covariance between any two of the customers. Additionally, Theorem 6.4 presents the non-intuitive result that $\text{Cov}(T_1, T_2) = \text{Cov}(T_1, T_{1000})$. The correlation decreases with increasing lag, however, as expected due to the diminishing effect of the intermediate customer sojourn times reflected in the denominator of the defining formula of the correlation. Theorem 6.4's results are confirmed by the Monte Carlo simulation in Appendix M and the Maple code used to compute the exact covariance values between two customers is listed in Appendix N.

6.8 Conclusions

Previous transient analysis results for the $M/M/1$ and $M/M/s$ queues have been combined with the functionality of the Maple computational engine (and subsequently APPL) to develop both symbolic and numeric exact sojourn time PDFs that can be manipulated to compute and study various measures of performance. A complete variance–covariance matrix for the first $n = 10$ customers and varying traffic intensity is calculated, illustrating this approach's ability to determine the joint PDF between two customer sojourn times. The results offer a framework for describing how the well-known $M/M/s$ queue steady-state results occur as the queue progresses toward steady-state. When possible, results are checked against corresponding Monte Carlo simulation and/or previous literature.

Chapter 7

Conclusions and Further Work

This dissertation presents a variety of applications in stochastic operations research, where each application requires the use of a computational engine. Without the computational engine, the results are either intractable or overly tedious to compute. These applications contribute to and extend the current literature for their respective fields. Additionally, the applications offer insight into other problems found to be similar in structure.

Harnessing the computational power available today and implementing it effectively requires the researcher to fundamentally understand the software and/or computational engine utilized as well as the first-principles theory of the problem at hand. The algorithms and associated code generally stem from a first-principles approach to the problem. As with most versions of coded algorithms, computational complexity is almost always of interest. As mentioned in the introduction, many other languages and/or software packages could have been utilized to achieve the same results, some in a faster and more efficient manner. Additionally, the author recognizes that the code, as it is written, might not be the most efficient means available. However, the algorithms and their respective code are meant to be straightforward in design and simplicity. Additionally, the author has made an effort to present procedures easily implemented by minimizing necessary syntax and setup work for the user. Where possible, computed results are available symbolically, using generic coding.

Substantiating results to problems addressed here is difficult. Most of the solutions are returned exactly, and the problems as described are overly tedious to calculate with pencil and paper. Therefore, careful attention is paid to substantiating results. The best method for verification is Monte Carlo simulation. In all cases, Monte Carlo simulation was used for multiple runs, each run with a corresponding high number of replications, to create a confidence interval for the exact solution. The problem complexity sometimes required these simulations to run for extended time periods, where CPU time for the Monte Carlo estimate exceeded CPU time to achieve the exact solution. Additionally, when variate generation methods were compared (Chapters 2 and 3), a Monte Carlo simulation was the best tool for conducting the study in the presence of multiple population distributions.

As an example of verifying previous results, consider the work in Chapter 6. Much of the work completed in this chapter was motivated by a course taken in Computational Probability where the monograph used for the course was Drew et al. (2007), explaining the use of Maple and APPL throughout. It is because of APPL that Chapter 6 of the dissertation became possible, where, by framing $M/M/1$ transient analysis queueing as a new class of computational probability problems, along with applying propositions made by Kelton and Law (1985), led to exact sojourn time distributions and sojourn time covariance calculations. Though the propositions have been known for 23 years, an application using them to calculate a distribution and subsequently measures of performance has eluded researchers. The procedures that this dissertation adds to APPL's suite of available tools furthers its capability by embarking on bivariate probability distribution functions. I hope the addition of these procedures aids in revolutionizing the field of computational probability, and its influence in research and education.

I now summarize the applications presented beginning with Chapter 2. Given an observed univariate data set assumed to come from an unknown continuous population distribution, generating variates for simulation from the piecewise-linear CDF created by connecting the steps created by the empirical distribution function is not novel. However, by doing so, the variance of the piecewise-linear estimator is always less than the unbiased sample variance of the observed data. Chapter 2 corrects the estimator by stretching and shifting the observed data such that the mean and variance of the estimator match the mean and variance of the data set, improving the quality of the variates produced by the estimator. Certain types of data might be inappropriate for stretching and shifting. For example, consider data arising from a service-time distribution, where the mean of the distribution is close to zero. Stretching and shifting such data might result in an impossible negative service time. To overcome this problem, Chapter 2 also offers an alternative method to match the mean and variance that does not affect the location of the observed data on the real line. This second method assigns appropriate weights to the data values that result in an estimator whose mean and variance equal the mean and variance of the sample data. The weights are are the solution to a nonlinear optimization program. Using either method bypasses the time-consuming and often arbitrary process of density estimation.

Chapter 3 extends the method of using a piecewise-linear CDF for variate generation to a two-dimensional random vector. The method presented is completely nonparametric and includes several examples showcasing its ability to effectively represent bivariate distributions with multiple modes. The method is a synchronized variate generation algorithm requiring only an observed bivariate data set from the user. The method used for the first variate in the two-dimensional random vector is produced exactly as outlined in Chapter 2. Using the first variate as a reference point, select data values from the sample data are collected to form a second piecewise-linear CDF conditioned on the value of the first variate. The second variate is produced via inversion from a conditional piecewise-linear CDF. Because each two-dimensional random vector produced requires the creation of a conditional weighted piecewise-linear CDF, this method is slower than its main competitor in the literature, kernel density estimation. However, while kernel density estimation can produce impossible variates in certain applications, the bivariate variate generation algorithm does not. Extensive comparisons are conducted, and results are provided in tabular and graphic form.

Select control chart constants for non-normal sampling are derived in Chapter 4. These are derived in symbolic form for the normal distribution as well as select non-normal distributions. APPL is used exclusively for the calculations in this chapter. The constants denoted as d_2, d_3, c_4, and c_5 are bias correction factors, where d_2 and d_3 correct for the mean and standard deviation of the sample range, and c_4 and c_5 correct for the mean of the sample standard deviation and its standard error. Although the constants associated with the normal distribution have been shown to be robust for non-normal processes, there can be substantial differences in control chart constants as shown in Tables 4.1 and 4.2. Conceivably, if an engineer knows enough about a process to warrant that the random variable of interest is non-normal in distribution, then he should be confident in estimating necessary parameters and applying the appropriate non-normal control chart constants. In the situation where estimated distribution parameters are required to derive the control chart constants, tabulated results are available up to reasonable sample sizes.

In Chapter 5, the KS goodness-of-fit test is offered as an alternative to the more traditional chi-square goodness-of-fit testing on whether leading digits of sample data conform to Benford's law. Before conducting the test, the data is transformed, where the data represented by the continuous random variable T is transformed as $Z = \log_{10} T \bmod 1$. The derivation shows that testing whether the leading digit of T conforms to Benford's distribution is equivalent to testing whether $Z \sim U(0, 1)$. Given this result, the KS test is appropriate (and exact) in testing whether $Z \sim U(0, 1)$. Since the test is exact for any sample size, the "rule of thumb" commonly suggested for chi-square goodness-of-fit testing (in testing for Benford, this was $n > 109$) is no longer necessary. The power curves for each test were plotted for four select non-Benford distributions. The results of the comparison suggest that the KS goodness-of-fit test should be used instead of the chi-square goodness-of-fit test.

In the final topic considered in the dissertation, transient queueing analysis is explored for the $M/M/s$ queue. The chapter begins by providing a background of the queueing discipline, along with an explanation of how transient analysis differs from steady-state analysis. Complete specification of customer sojourn time distribution are developed for the nth customer when $k \geq 0$ customers are present in the system at time zero. Examples are provided where, using the sojourn time distribution, several measures of performance are examined for varying customer numbers. The customer sojourn time distribution is extended to create bivariate sojourn time distributions for pairs of customers, allowing calculations of exact correlations and covariances between customers. A complete variance–covariance matrix is provided for $n = 10$ customers for varying traffic intensities. The APPL and Maple code written is made available in the appendix as well as online at `http://www.math.wm.edu/~leemis/QueueAPPL.txt`. These procedures provide a framework for investigating the behavior of the $M/M/s$ queue as it evolves toward steady-state.

As an extension of the work already completed in Chapter 2, where a piecewise-linear cumulative distribution function was formed by connecting points on a given empirical cumulative distribution function, I propose to extend the the work to the Kaplan–Meier estimator. Specifically, I will apply the univariate nonparametric variate generation work to generating variates from the Kaplan–Meier estimator for right-censored data in mathematical reliability theory. In the current literature, variates are generated from this estimate in a manner similar to bootstrapping (i.e., sampling with replacement). It is my intent to produce an algorithm that connects knot points along the Kaplan–Meier estimate with continuous segments allowing interpolation for variates from an estimator that shares the mean and variance of the observed lifetime data. Creating this estimator requires that the lifetime data be stretched and shifted so the resluting estimator's moments match those of the lifetime data.

In Chapter 3, five interesting potential areas of further work for the proposed algorithm are immediately evident. The first deals with studying how changes in the weighting function, w_k, of the interior points, $\boldsymbol{x}_A$, affect the resulting random pairs produced by the algorithm. The second area concerns the use of nonconvex hulls that allow for "dents" in the support. The third area concerns a two-dimensional extension of Marsaglia's tail algorithm. The fourth area concerns generation speed. The current algorithm generates the first element of the bivariate pair quickly and the second element slowly. The setup portion of the algorithm can be modified so as to generate both variates quickly by storing a set of conditional PDFs. The fifth area concerns how a similar algorithm might be extended to higher dimensions.

One area of further study for Chapter 4 might exist in searching for relationships analogous to (4.7), which depend only on the single parameter n. Burr (1968) presents a strong case for normal sampling applications; however, Wheeler (2000) notes a trough in Burr's skewness versus kurtosis plot (especially in U-shaped distributions) where using normal-sampling-based control chart constants would severely misrepresent the population.

When testing conformance to Benford's law in Chapter 5, future work includes examining whether the KS test is appropriate for populations conforming to Stigler's law as well as mixtures of other leading digit distributions. This potentially allows for testing more classes of distributions with a test that is exact for any sample size.

The first-principles derivation displayed in Chapter 6 suggests a possibility for future research would be to apply the approaches provided in this work to a $G/G/1$ and perhaps even the more general $G/G/s$ queue. Even though symbolic results might prove impossible, later versions of computational engines have improved in numeric methods, increasing the likelihood of achieving solutions. Additionally, making use of other computational formulae (such as Hagwood (2009)) may offer significant time savings in calculations and is another interesting avenue for future work.

Appendix A

Relationship to the Trapezoidal and Simpson's Rule

Suppose that $F = F_X\colon \mathbb{R} \to [0,1]$ is an unknown *continuous* cumulative distribution function (CDF) and $X_1, X_2, \ldots, X_n$ are i.i.d. random variables with this distribution. Let

$$x_{(1)} \leq x_{(2)} \leq \cdots \leq x_{(n)} \tag{A.1}$$

denote the particular values obtained in a given random sample, sorted into weakly increasing order. Our goal is to use this sample to estimate F, which will then be used to simulate further observations. (We additionally assume that the support of the population is positive for simplicity. If the lower bound of the support happens to be a finite negative value, the results given in this appendix can be achieved by shifting the data values and adjusting the associated moments.)

For convenience, assume that F is strictly increasing on some (unknown) interval of possible values $[a,b]$. Thus, $F\colon [a,b] \to [0,1]$ is an invertible function with inverse $F^{-1}\colon [0,1] \to [a,b]$. Letting $U \sim U(0,1)$, the probability integral transformation (Fishman, 2006, page 77) states that X and each X_i has the same distribution as

$F^{-1}(U)$. In particular,

$$E[X^k] = \int_0^1 F^{-1}(u)^k \, du.$$

Furthermore, to simulate random observations from the distribution of X, we need only use a random number generator to generate random numbers $u \in [0,1]$, and then compute $F^{-1}(u)$.

Let $y_{(i)} = (i-1)/(n-1)$ for $i = 1, 2, \ldots, n$. Given the input data (A.1), symmetry suggests that we estimate F by a piecewise-linear function for which $F_0(x_{(i)}) = y_{(i)}$ for $i = 1, 2, \ldots, n$. This is equivalent to estimating F^{-1} by a piecewise-linear function F_0^{-1} such that $F_0^{-1}(y_{(i)}) = x_{(i)}$ for $i = 1, 2, \ldots, n$.

More generally, we might postulate that F^{-1} is some continuous function (not necessarily piecewise linear) such that $F^{-1}(y_{(i)}) = x_{(i)}$ for $i = 1, 2, \ldots, n$. We can then use numerical integration techniques to estimate integrals involving the unknown function F^{-1}. This is easy to do, since the $y_{(i)}$'s form a partition of $[0,1]$ into $n-1$ subintervals of equal length. For example, using the trapezoidal rule to estimate $E[X]$ gives

$$\begin{aligned}
E[X] &= E[F^{-1}(U)] \\
&= \int_0^1 F^{-1}(u) \, du \\
&\approx \frac{1-0}{2(n-1)} \left(F_0^{-1}(y_{(1)}) + 2F_0^{-1}(y_{(2)}) + 2F_0^{-1}(y_{(3)}) + \cdots + F_0^{-1}(y_{(n)}) \right) \\
&= \frac{x_{(1)} + 2x_{(2)} + 2x_{(3)} + \cdots + 2x_{(n-1)} + x_{(n)}}{2(n-1)}.
\end{aligned}$$

Of course, this is exactly the formula obtained by using a piecewise-linear approximation in Section 2.1. Similarly, the trapezoidal estimate of $E[X^2]$ is

$$E[X^2] = \int_0^1 F^{-1}(u)^2 \, du \approx \frac{x_{(1)}^2 + 2x_{(2)}^2 + \cdots + 2x_{(n-1)}^2 + x_{(n)}^2}{2(n-1)}.$$

We remark that this expression does not necessarily equal $\int_0^1 F_0^{-1}(u)^2 \, du$, but it is certainly one reasonable way to estimate $E[X^2]$. Note that both of our formulas give unbiased estimators for the mean and second moment of X, although these are not the usual unbiased estimators commonly employed in statistics.

The simplest approach to simulating observations from X is to use the piecewise-linear estimate F_0^{-1} for F^{-1}. One more advanced approach is to replace F_0^{-1} by some affine transformation $F_1^{-1} = cF_0^{-1} + d$, for suitable constants c, d. One way to proceed is to choose c and d so that $E[F_1^{-1}(U)]$ equals the sample mean of the $x_{(i)}$'s, and $\mathrm{Var}[F_1^{-1}(U)]$ equals the unbiased sample variance of the $x_{(i)}$'s. A related approach (which is a bit simpler computationally) is to choose c and d so that the trapezoidal estimates of $E[X]$ and $E[X^2]$ (computed with respect to F_1^{-1}) equal the corresponding sample moments (computed using the $x_{(i)}$'s). In more detail, let $m_1 = \sum_i x_{(i)}$, $m_2 = \sum_i x_{(i)}^2$,

$$t_1 = \frac{x_{(1)} + 2x_{(2)} + \cdots + 2x_{(n-1)} + x_{(n)}}{2(n-1)}, \qquad t_2 = \frac{x_{(1)}^2 + 2x_{(2)}^2 + \cdots + 2x_{(n-1)}^2 + x_{(n)}^2}{2(n-1)}.$$

Then we can choose c and d to satisfy

$$m_1 = ct_1 + d, \quad m_2 = c^2 t_2 + 2cdt_1 + d^2.$$

We then simulate random observations from X by generating random numbers $u \in [0, 1]$, and computing simulated values $cF_0^{-1}(u) + d$.

The preceding discussion suggests some tantalizing extensions. What if we used more advanced numerical integration techniques to estimate integrals involving the unknown function F^{-1}? For example, when $n-1$ is even, we could use Simpson's Rule to estimate $E[X]$ and $E[X^2]$, which amounts to using piecewise-quadratic estimates of the functions $F^{-1}(y)$ and $F^{-1}(y)^2$. This leads to formulas such as

$$E[X] = E[F^{-1}(U)] \approx \frac{x_{(1)} + 4x_{(2)} + 2x_{(3)} + 4x_{(4)} + \cdots + 4x_{(n-1)} + x_{(n)}}{3(n-1)}.$$

One could then try to modify the associated piecewise-quadratic functions by affine transformations to attain a closer match to the sample mean and unbiased sample variance.

Appendix B

Computing δ

The quadratic equation in δ in Section 2.2.1 is

$$\sum_{i=1}^{n-1} \frac{(x'_{(i)})^2 + x'_{(i)}x'_{(i+1)} + (x'_{(i+1)})^2}{3(n-1)} - \left[\frac{x'_{(1)} + 2\sum_{i=2}^{n-1} x'_{(i)} + x'_{(n)}}{2(n-1)}\right]^2 = s^2,$$

where $x'_{(i)} = x_{(1)} - \delta + w\sum_{j=1}^{i-1} g'_j, i = 1, 2, \ldots, n$. This appendix contains the algebra and an associated S-Plus/R function to solve this equation.

First, simplify the expression for $x'_{(i)}$ as

$$\begin{aligned} x'_{(i)} &= x_{(1)} - \delta + \left(x_{(n)} - x_{(1)} + 2\delta\right)\sum_{j=1}^{i-1} \frac{x_{(i+1)} - x_{(i)}}{x_{(n)} - x_{(1)}} \\ &= x_{(1)} - \delta + \frac{x_{(n)} - x_{(1)} + 2\delta}{x_{(n)} - x_{(1)}}\left(x_{(i)} - x_{(1)}\right) \\ &= x_{(i)} + \left[\frac{2x_{(i)} - x_{(n)} - x_{(1)}}{x_{(n)} - x_{(1)}}\right]\delta. \end{aligned}$$

Define $r_{(i)}$ as the portion of this equation in the brackets:

$$r_{(i)} = \frac{2x_{(i)} - x_{(n)} - x_{(1)}}{x_{(n)} - x_{(1)}},$$

for $i = 1, 2, \ldots, n$. Thus $x'_{(i)}$ can be written more compactly as

$$x'_{(i)} = x_{(i)} + r_{(i)}\delta.$$

Returning to the quadratic equation and replacing $x'_{(i)}$ with $x_{(i)} + r_{(i)}\delta$ yields:

$$\sum_{i=1}^{n-1} \frac{(x_{(i)} + r_{(i)}\delta)^2 + (x_{(i)} + r_{(i)}\delta)(x_{(i+1)} + r_{(i+1)}\delta) + (x_{(i+1)} + r_{(i+1)}\delta)^2}{3(n-1)}$$

$$- \left[\frac{x_{(1)} + 2\sum_{i=2}^{n-1}(x_{(i)} + r_{(i)}\delta) + x_{(n)}}{2(n-1)}\right]^2 - s^2 = 0.$$

Expanding this quadratic equation in δ in the form $a\delta^2 + b\delta + c = 0$ and collecting terms yields the following expressions for a, b, and c:

$$a = \sum_{i=1}^{n-1} \frac{r_{(i)}^2}{3(n-1)} + \sum_{i=1}^{n-1} \frac{r_{(i)}r_{(i+1)}}{3(n-1)} + \sum_{i=1}^{n-1} \frac{r_{(i+1)}^2}{3(n-1)} - \frac{1}{(n-1)^2}\left(\sum_{i=2}^{n-1} r_{(i)}\right)^2$$

$$b = \sum_{i=1}^{n-1} \frac{2x_{(i)}r_{(i)}}{3(n-1)} + \sum_{i=1}^{n-1} \frac{x_{(i)}r_{(i+1)} + x_{(i+1)}r_{(i)}}{3(n-1)} + \sum_{i=1}^{n-1} \frac{2x_{(i+1)}r_{(i+1)}}{3(n-1)}$$
$$- \frac{1}{(n-1)^2}\left[x_{(1)} + 2\sum_{i=2}^{n-1} x_{(i)} + x_{(n)}\right]\sum_{i=2}^{n-1} r_{(i)}$$

$$c = \sum_{i=1}^{n-1} \frac{x_{(i)}^2}{3(n-1)} + \sum_{i=1}^{n-1} \frac{x_{(i)}x_{(i+1)}}{3(n-1)} + \sum_{i=1}^{n-1} \frac{x_{(i+1)}^2}{3(n-1)} - \frac{\left(x_{(1)} + 2\sum_{i=2}^{n-1} x_{(i)} + x_{(n)}\right)^2}{4(n-1)^2} - s^2$$

Given the values of a, b, and c, the positive root of the quadratic equation is given by $\delta = \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$. The values of δ and $x''_{(1)}, x''_{(2)}, \ldots, x''_{(n)}$ can be calculated in S-Plus using the function mm (for "matching moments") given below.

```
mm <- function(x) {
  x    <- sort(x)
  n    <- length(x)
  xbar <- mean(x)
  xvar <- var(x)
  r    <- (2 * x - x[n] - x[1]) / (x[n] - x[1])
  rlo  <- r[1:(n - 1)]
  rhi  <- r[2:n]
  rmid <- r[2:(n - 1)]

  xlo  <- x[1:(n - 1)]
  xhi  <- x[2:n]
  xmid <- x[2:(n - 1)]

  aa   <- 1 / (3 * (n - 1)) * (sum(rlo * rlo) + sum(rlo * rhi) +
          sum(rhi * rhi)) - 1 / (n - 1) ^ 2 * (sum(rmid) ^ 2)
  bb   <- 1 / (3 * (n - 1)) * (sum(2 * xlo * rlo) + sum(xlo *
          rhi + xhi * rlo) + sum(2 * xhi * rhi)) - 1 /
          (n - 1) ^ 2 * sum(rmid) * (x[1] + x[n] + 2 * sum(xmid))
  cc   <- 1 / (3 * (n - 1)) * (sum(xlo ^ 2) + sum(xlo * xhi) +
          sum(xhi ^ 2)) - 1 / (4 * (n - 1) ^ 2) * ((x[1] + x[n] +
          2 * sum(xmid)) ^ 2) - xvar

  del  <- (-bb + sqrt(bb ^ 2 - 4 * aa * cc)) / (2 * aa)
  xp   <- x + r * del
  xpp  <- xp - ((sum(xp) - xp[1] / 2 - xp[n] / 2) / (n - 1) - xbar)
  xpp
}
```

As expected, this function returns the vector of values, namely,

```
-0.1347206  1.1080189  4.8362375  7.3217166  8.5644561  9.8071956
```

from Example 3 when called with

```
mm(c(1, 2, 5, 7, 8, 9))
```

This function can be downloaded from `www.math.wm.edu/~leemis/2009mm.code`.

Appendix C

Nonparametric Bivariate Generator

The R/S-Plus code below contains all the elements necessary to generate random bivariate pairs given the $\boldsymbol{x}$ and $\boldsymbol{y}$ vectors consisting of the observed data using the algorithm described in Chapter 3. The code is separated into three portions, setup, generation, and the main program. Indentation denotes nesting.

C.1 xpwl(x) and ywtpwl(xgen)

```
# OPTIONAL MOMENT MATCHING SETUP PORTION
# xnew <- mm(x)
# ynew <- mm(y)

orderxnew  <- order(xnew)
xnewlength <- length(xnew)

x <- xnew[orderxnew]
y <- ynew[orderxnew]

hullindex <- chull(x,y)
m <- length(hullindex)
```

```
xhull <- x[hullindex]
yhull <- y[hullindex]
hullorder <- order(xhull,yhull)
indexmin <- hullorder[1]
indexmax <- hullorder[m]

# determine the line separating the upper and lower hull
slope <- (yhull[indexmax] - yhull[indexmin]) / (xhull[indexmax]
         - xhull[indexmin])
intercept <- yhull[indexmax] - slope * xhull[indexmax]
count <- 0

# find length (segments) of upper and lower hulls
count <- length(which(yhull[hullorder] > slope * xhull[hullorder]
         + intercept))

# VARIATE GENERATION FUNCTIONS

# generate x from the piecewise-linear CDF from original
# (or moment matched) x vector

xpwl <- function(x) {
   u <- runif(1)
   i <- ceiling((xnewlength - 1) * u)
   x[i] + ((xnewlength - 1) * u - (i - 1)) * (x[i + 1] - x[i])
}

# generate y from the weighted piecewise-linear CDF created by
# conditioning on the x value generated

ywtpwl <- function(xgen) {

# find segments of hull lower and upper intersection with xgen,
# determine intersecting y values

   for (i in 1:length(upperx)) {
      if ((xgen >= upperx[i]) && (xgen <= upperx[i + 1])) {
      upperslope <- (uppery[i] - uppery[i+1]) / (upperx[i] -
                   upperx[i + 1])
      upperint <- uppery[i + 1] - upperslope * upperx[i + 1]
      ymax <- upperslope * xgen + upperint
      }
```

```
  }

  for (i in 1:length(lowerx)) {
     if ((xgen >= lowerx[i]) && (xgen <= lowerx[i + 1])) {
        lowerslope <- (lowery[i] - lowery[i + 1]) / (lowerx[i]
                      - lowerx[i + 1])
        lowerint <- lowery[i + 1] - lowerslope * lowerx[i + 1]
        ymin <- lowerslope * xgen + lowerint
     }
  }

# collect y values between ymin and ymax forming the set A
  j <- 0
  ybetweenindex <- 0
  for (i in 1:xnewlength) {
     if (y[i] <= ymax & y[i] >= ymin) {
        j <- j + 1
        ybetweenindex[j] <- i
     }
  }

# create x and y vectors for interior points, augment
# with ymin, ymax
  ybetween <- y[ybetweenindex]
  xbetween <- x[ybetweenindex]
  ybetweenorder <- order(ybetween)
  yvec <- c(ymin, ybetween[ybetweenorder], ymax)
  xvec <- c(xgen, xbetween[ybetweenorder], xgen)

# weight y values by distance from xgen, w(i) = 1 / (1 +
# ((x(i) - xgen) / sqrt(var(xvec))) ^ 2)
  yweight <- 0
  for (i in 1:length(yvec)) {
     yweight[i] <- 1 / (1 + ((xvec[i] - xgen) /
                   sqrt(var(xvec))) ^ 2)
  }

# normalize weights
  ynmwt <- 0
  for (i in 1:length(yvec)) {
     ynmwt[i] <- yweight[i] / sum(yweight)
  }
```

```
 # find new y knot points of the weighted piecewise-linear CDF
   yknots <- matrix(0:0, length(yvec))
   yknots[1] <- 0
   for (i in 2:length(yvec)) {
      yknots[i] <- sum(ynmwt[1:(i - 1)]) + (i - 1) * (ynmwt[i])
                   / (length(yvec) - 1)
   }

   # generate y value pwl from knot point y values
   u1 <- runif(1)
   i <- 1
   while (u1 > yknots[i + 1]) {
      i <- i + 1
   }
   yvec[i] + (u1 - yknots[i]) * (yvec[i + 1] - yvec[i]) /
   (yknots[i + 1] - yknots[i])
}

# MAIN PROGRAM

# set N to the desired number of random variates here
# Generated <- matrix(0:0, N, 2) collects the resulting
# random variate pairs

for (i in 1:N) {
   xgen <- xpwl(x)
   ygen <- ywtpwl(xgen)
   Generated[i, 1] <- xgen
   Generated[i, 2] <- ygen
}
```

Appendix D

Creating the Sojourn Time Distribution

The following procedures do not exist in APPL and were written to accomplish the goals outlined in Chapter 6. They make internal use of other APPL procedures and are intended to become part of the procedures offered in the APPL suite of software.

D.1 Queue(X, Y, n, k, s)

```
# Queue(X, Y, n, k, s)
# ---------------------------------------------------
# Computes the sojourn time distribution of the nth
# customer in an M/M/s queue, given k customers are
# in the system at time 0.  Queue calls the
# subprocedure MMsQprob(n, k, s) (and subsequently
# calls Q(n, i, k, s)) which recursively calculates
# the required probabilities of the nth customer
# seeing exactly i customers, including himself, in
# the system upon arrival for i = 1 to n + k
# customers.  Calculations are based on algorithms
# provided in Kelton and Law (1985).  Calls the
# subprocedure BuildDist(X, Y, n, k, s) which builds
# the conditional sojourn time distribution for
# i = 1 customer to i = n + k customers in the
# system. Queue mixes the probabilities with the
# conditional sojourn time distributions to return
# the exact PDF of the sojourn time in the APPL
# list-of-lists format.  Requires the
# subprocedures mentioned above along with the APPL
# software.  The exponential arrival and service
# random variables must be defined in the APPL
# format. The procedure call is
# Queue(X, Y, n, k, s), where X is the arrival time
# distribution, Y is the service time distribution,
# n is the customer of interest, k is the number of
# customers in the system at time 0, and s is the
# number of identical parallel servers.  Both X and
# Y must be exponential random variables in the
# APPL list-of-lists format.
#
# Name            : Queue.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 09/11/08
# ---------------------------------------------------
Queue := proc(X, Y, n, k, s)
   global rho;
   local i :: integer, lst :: list, TIS :: list;
   rho := 1 / Mean(X) / (s * 1 / Mean(Y));
   lst := BuildDist(X, Y, n, k, s);
   TIS := Mixture(MMsQprob(n, k, s), lst);
   return TIS;
end:
```

D.2 MMSQprob(n, k, s)

```
# MMSQprob(n, k, s)
# --------------------------------------------------
# Computes Pk(n, i)'s for an M/M/s queue, which is
# the probability that customer n will see i
# customers in the system including himself at time Tn
# with k customers initially in the system at time 0.
# Calls the subprocedure Q(n, i, k, s) which
# recursively calculates the required probabilities
# using the algorithms provided in Kelton and Law
# (1985). The procedure returns the ordered
# probabilities for i = 1 customer (an empty queue)
# to i = n + k customers in a list.  Note that the
# parameter rho for an M/M/s queue is
# rho = lambda / (s * mu).
#
# Name            : MMsQprob.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 09/03/08
# --------------------------------------------------
MMsQprob := proc(n, k, s)
   local i :: integer, lst :: list;
   lst := [];
   for i from 1 to n + k do
      lst := [op(lst), Q(n, i, k, s)];
   od;
   return lst;
end;
```

D.3 Q(n, i, k, s)

```
# Q(n, i, k, s)
# ---------------------------------------------------
# Computes the single probability Pk(n, i) for an
# M/M/s queue recursively according to the algorithms
# provided in Kelton and Law (1985).
# Name            : Q.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 09/03/08
# ---------------------------------------------------
Q := proc(n, i, k, s)
   option remember;
   global rho;
   local p, j :: integer, h :: integer;
   if (k >= 1) and (i = k + n) then
      if k >= s then p := (rho / (rho + 1)) ^ n
      elif k + n <= s then
         p := rho ^ n / (mul(rho + (k + j - 1) / s,
              j = 1 .. n))
      elif (k < s) and (s < k + n) then
         p := rho ^ n / ((rho + 1) ^ (n - s + k) *
              (mul(rho + (k + j - 1) / s,
              j = 1 .. s - k)))
      fi;
   fi;
   if (k = 0) and (i = n) then
      if n <= s then
         p := rho ^ n  / mul(rho + (j - 1) / s,
              j = 1 .. n)
      elif n > s then
         p := rho ^ n / ((rho + 1) ^ (n - s) *
              mul(rho + (j - 1) / s, j = 1 .. s))
      fi;
   fi;
   if i = 1 then
      p := 1 - add(Q(n, j, k, s), j = 2 .. n + k)
   fi;
   if (k >= 1) and (i >= 2) and (i <= k) and (n = 1) then
      if k <= s then
```

```
         p := rho / (rho + (i - 1) / s) * mul(1 - rho
              / (rho + (k - j + 1) / s),
              j = 1 .. k - i + 1)
      elif (k > s) and (i > s) then
         p := rho / (rho + 1) ^ (k - i + 2)
      elif (i <= s) and (s < k) then
         p := rho / ((rho + 1) ^ (k - s + 1) * (rho +
              (i - 1) / s)) * mul(1 - rho / (rho +
              (s - j) / s), j = 1 .. s - i)
      fi;
   fi;
   if (n >= 2) and (i >= 2) and (i <= k + n - 1) then
      if i > s then
         p := rho / (rho + 1) * add((1 / (rho + 1) ^
              (j - i + 1) * Q(n - 1, j, k, s)),
              j = i - 1..k + n - 1)
      elif i <= s then
         p := rho / (rho + (i - 1) / s) *
              (add((mul(1 - rho / (rho + (j - h + 1)
              / s), h = 1..j - i + 1)) * Q(n - 1, j,
              k, s), j = i - 1..s - 1) + product(1 -
              rho / (rho + (s - h) / s), h = 1 .. s -
              i) * add((1 / (rho + 1)) ^ (j - s + 1)
              * Q(n - 1, j, k, s), j = s .. k +
              n - 1));
      fi;
   fi;
   return p;
end:
```

D.4 BuildDist(X, Y, n, k, s)

```
# BuildDist(X, Y, n, k, s)
# -----------------------------------------------
# Creates the appropriate conditional sojourn time
# distribution for each case where a customer arrives
# to find i = 1 to i = n + k customers present,
# including himself, in an M/M/s queue with k customers
# intially present. The procedure call is
# BuildDist(X, Y, n, k, s), where X is the arrival time
# distribution, Y is the service time distribution, n
# is the customer number of interest, k is the number
# of customers in the system at time 0, and s is the
# number of identical parallel servers.  Both X and Y
# must be exponential random variables in the APPL
# list-of-lists format.
#
# Name            : BuildDist.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 09/03/08
# -----------------------------------------------
BuildDist := proc(X, Y, n, k, s)
   local i :: integer, lst :: list;
   lst := [];
   for i from 1 to n + k do
     if s = 1  then
       lst := [op(lst), ErlangRV(1 / Mean(Y), i)]
     else
       if (i <= s) or (s > n + k) then
          lst := [op(lst), Y];
       else
          lst := [op(lst), Convolution(ErlangRV(s *
                 1 / Mean(Y), i - s), Y)];
       fi;
     fi;
   od;
   return lst;
 end;
```

Appendix E

Average Delay and Service for Percentile Comparison

```
/* ----------------------------------------------------------
 * This program alters ssq1.c from Leemis and Park (2005)
 * to capture the average delay and service times for an
 * M/M/1 queue with an arrival rate = 1 and a service
 * rate = 10/9 for the third customer, given the queue is
 * empty and idle at time T = 0.  The program also writes
 * 10,000,000 service times to use as an empirical CDF for
 * comparing the 99th percentile to the exact percentile
 * as provided by the APPL procedure Queue.
 *
 * Name              : mm1.c
 * Author            : Billy Kaczynski
 * Language          : ANSI C
 * Latest Revision   : 9-10-08
 * ----------------------------------------------------------
 */

#include <stdio.h>
#include <math.h>
#include "rng.h"
#define LAST         4L      /* number of jobs processed */
#define START        0.0     /* initial time             */
FILE * fptr;
```

```
   double Exponential(double m)
/* ---------------------------------------------------
 * generate an Exponential random variate, use m > 0.0
 * ---------------------------------------------------
 */
{
  return (-m * log(1.0 - Random()));
}

   double GetArrival(void)
/* ------------------------------
 * generate the next arrival time
 * ------------------------------
 */
{
  static double arrival = START;
  arrival += Exponential(1.0);
  return (arrival);
}

   double GetService(void)
/* ------------------------------
 * generate the next service time
 * ------------------------------
 */
{
  return (Exponential(0.9));
}

  int main(void)
{
  long   index     = 0;            /* job index             */
  long   i         = 10000000;
  long   t         = 0;
  double arrival   = START;        /* time of arrival       */
  double delay;                    /* delay in queue        */
  double service;                  /* service time          */
  double wait;                     /* delay + service       */
  double departure = START;        /* time of departure     */
  struct {                         /* sum of ...            */
    double delay;                  /*   delay times         */
    double wait;                      /*   wait times         */
    double service;                   /*   service times      */
    double interarrival;              /*   interarrival times */
  } sum = {0.0, 0.0, 0.0};

  PutSeed(123456789);
  fptr = fopen("data.dat", "w");
  for (t = 0; t < i; t++) {
    index = 0;
    arrival = START;
    departure = START;
  while (index < LAST) {
    index++;
    arrival       = GetArrival();
    if (arrival < departure)
      delay       = departure - arrival; /*delay in queue*/
    else
      delay       = 0.0;                 /* no delay     */
    service       = GetService();
    wait          = delay + service;
    departure     = arrival + wait;  /*time of departure */

    if (index == 3) {
    sum.delay   += delay;
    sum.wait    += wait;
    sum.service += service;
    fprintf(fptr, "%7.5lf\n", wait); }
  }
  sum.interarrival = arrival - START;
  }
  fclose(fptr);
  printf("\nfor the %ldrd job\n", index - 1);
  printf("   average wait ............ = %6.5f\n",
            sum.wait / i);
  printf("   average delay .......... = %6.5f\n",
            sum.delay / i);
  printf("   average service time .... = %6.5f\n",
            sum.service / i);
    return (0);
}
```

Appendix F

Discrete-Event Simulations for Customers 1 and 2

The following simulations are written for R/S-Plus providing the measures of performance for approaches 1 and 2 in Section 6.5.1. The time to execute the simulation is negligible. Each code segment has been vectorized in order to take advantage of R/S-Plus' efficiency in manipulating vectors.

```
# ------------------------------------------------
# Next-event discrete-event simulation for customers
# 1 and 2 to calculate their covariance and
# correlation.
# Name            : approach1.txt
# Author          : Billy Kaczynski
# Language        : R/S-Plus
# Latest Revision : 10/29/08
# ------------------------------------------------

# Approach 1:

N <- 10000000
lambda <- 1
mu <- 2
```

```
t1   <- rexp(N, mu)
a2   <- rexp(N, lambda)
c1   <- which(a2 > t1)
c2   <- which(t1 > a2)
t1c1 <- t1[c1]
t1c2 <- t1[c2]
Y    <- t1[c2] - a2[c2]
t2c1 <- rexp(length(c1), mu)
t2c2 <- rexp(length(Y), mu) + Y
t1   <- c(t1c1, t1c2)
t2   <- c(t2c1, t2c2)

mean(t1)
var(t1)
mean(t2)
var(t2)
mean(Y)
var(Y)
mean(t2c2)
var(t2c2)
cov(t1, t2)
cor(t1, t2)
```

```
# ---------------------------------------------------
# Conditional discrete-event simulation for customers
# 1 and 2 to calculate their covariance and
# correlation.
# Name            : approach2.txt
# Author          : Billy Kaczynski
# Language        : R/S-Plus
# Latest Revision : 10/29/08
# ---------------------------------------------------

# Approach 2:

N <- 10000000
U <- runif(N)
lambda <- 1
mu <- 2
p <- lambda / (lambda + mu)

t1   <- rexp(N, lambda + mu)
t2   <- rexp(N, mu)
c1   <- which(U > p)
c2   <- which(U < p)
Y    <- rexp(length(c2), mu)
t1c1 <- t1[c1]
t1c2 <- t1[c2] + Y
t2c1 <- t2[c1]
t2c2 <- t2[c2] + Y
t1   <- c(t1c1, t1c2)
t2   <- c(t2c1, t2c2)

mean(t1)
var(t1)
mean(t2)
var(t2)
mean(Y)
var(Y)
mean(t2c2)
var(t2c2)
cov(t1, t2)
cor(t1, t2)
```

Appendix G

Derivation of the Trivariate Sojourn Time Distribution

The trivariate joint probability distribution of the sojourn times for customers 1, 2, and 3 in an initially empty and idle $M/M/1$ queue is derived below. The approach uses first principles for each of the five possible arrival/departure ordering sequences, along with a geometric description of the more complicated cases. The cases are ordered by increasing complexity as A, B, C, D, and E, with A being the simplest case, where the first three sojourn times are independent. Cases B and C each possess one independent sojourn time and two dependent sojourn times. Cases D and E are the most complicated cases, where dependence occurs between all three customers.

Let T_i be the sojourn time for customer $i = 1, 2, 3$ in an $M/M/1$ queue with arrival rate λ and service rate μ. Define $F_A(t_1, t_2, t_3)$ and $f_A(t_1, t_2, t_3)$ as the joint CDF and PDF respectively for case A for the first three customers. The notation for cases B through E follow accordingly. Define the probabilities $p_A, p_B, \ldots, p_E$ as the probability that cases A through E occur respectively. The case matrix, C, is

$$C = \begin{bmatrix} 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix},$$

where rows 1 through 5 of C correspond to cases A through E, with 1 indicating an arrival and -1 indicating a departure. The corresponding segment distribution matrix C' for $n = 3$ is

$$C' = \begin{bmatrix} 1 & 0 & 1 & 0 & 2 \\ 1 & 0 & 1 & 2 & 2 \\ 1 & 1 & 1 & 0 & 2 \\ 1 & 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 & 2 \end{bmatrix}.$$

There are three possible entries in C', corresponding to the distribution for each successive segment:

- exponential($\lambda + \mu$), which is indicated by a 1
- exponential(μ), which is indicated by a 2
- no distribution as a result of an emptied system, which is indicated by a 0.

Case A. In case A, the first three customer sojourn times are independent. The joint CDF is therefore the product of the CDFs for each customer. The path segment distributions for T_1, T_2, T_3 are exponential($\lambda + \mu$), exponential($\lambda + \mu$), exponential(μ) respectively. Therefore,

$$F_A(t_1, t_2, t_3) = \left(1 - e^{-(\lambda+\mu)t_1}\right)\left(1 - e^{-(\lambda+\mu)t_2}\right)\left(1 - e^{-\mu t_3}\right) \qquad t_1, t_2, t_3 > 0.$$

Case A occurs with probability $p_A = \mu^2/(\lambda+\mu)^2$. Taking partial derivatives yields the joint PDF

$$f_A(t_1,t_2,t_3) = \mu\left(\lambda+\mu\right)^2 e^{-\lambda t_1-\mu t_1-\lambda t_2-\mu t_2-\mu t_3} \qquad t_1,t_2,t_3 > 0.$$

Case B. In case B, customer 1's sojourn time is independent of the sojourn times of customers 2 and 3. Customers 2 and 3 share a distribution segment whose duration is denoted by Y. The path segment distributions for T_1, T_2, T_3 are exponential$(\lambda+\mu)$, exponential$(\lambda+\mu)+Y$, $Y+$ exponential(μ), respectively, where $Y \sim$ exponential(μ). Because of the dependence occurring between customers 2 and 3, the value of Y cannot exceed either the value of T_2 or of T_3. Thus the interval over which y varies depends on the relative sojourn time possibilities $t_2 < t_3$ and $t_3 < t_2$. Conditioning on Y, the joint CDF for case B is

$$F_B(t_1,t_2,t_3) = \begin{cases} \displaystyle\int_0^{t_2} \left(1-e^{-(\lambda+\mu)t_1}\right)\left(1-e^{-(\lambda+\mu)(t_2-y)}\right) & \\ \qquad \left(1-e^{-\mu(t_3-y)}\right)\mu e^{-\mu y}dy & t_2 < t_3 \\ \displaystyle\int_0^{t_3} \left(1-e^{-(\lambda+\mu)t_1}\right)\left(1-e^{-(\lambda+\mu)(t_2-y)}\right) & \\ \qquad \left(1-e^{-\mu(t_3-y)}\right)\mu e^{-\mu y}dy & t_3 < t_2. \end{cases} \tag{G.1}$$

Case B occurs with probability $p_B = \lambda\mu/(\lambda+\mu)^2$. Taking partial derivatives yields the joint PDF

$$f_B(t_1,t_2,t_3) = \begin{cases} \mu^2\left(-\lambda-\mu+e^{t_2(\lambda+\mu)}\lambda+\mu e^{t_2(\lambda+\mu)}\right)e^{-t_1\lambda-t_1\mu-\lambda t_2-\mu t_2-\mu t_3} & t_2 < t_3 \\ \mu^2\left(\lambda+\mu\right)\left(e^{t_3(\lambda+\mu)}-1\right)e^{-\lambda t_1-\mu t_1-\lambda t_2-\mu t_2-\mu t_3} & t_3 < t_2. \end{cases}$$

Case C. This case is analogous to case B except that customer 3's sojourn time is independent and customers 1 and 2 share the distribution segment whose duration is denoted by Y. The path segment distributions for T_1, T_2, T_3 are exponential$(\lambda+\mu)+Y$, $Y+$ exponential$(\lambda+\mu)$, exponential(μ), respectively, where $Y \sim$ exponential$(\lambda+\mu)$.

Conditioning on Y, the joint CDF, which is similar in structure to (G.1) for case B, is

$$F_C(t_1,t_2,t_3) = \begin{cases} \int_0^{t_1} \left(1-e^{-(\lambda+\mu)(t_1-y)}\right)\left(1-e^{-(\lambda+\mu)(t_2-y)}\right) \\ \qquad \left(1-e^{-\mu t_3}\right)(\lambda+\mu)\,e^{-(\lambda+\mu)y}dy & t_1<t_2 \\ \int_0^{t_2} \left(1-e^{-(\lambda+\mu)(t_1-y)}\right)\left(1-e^{-(\lambda+\mu)(t_2-y)}\right) \\ \qquad \left(1-e^{-\mu t_3}\right)(\lambda+\mu)\,e^{-(\lambda+\mu)y}dy & t_2<t_1. \end{cases} \tag{G.2}$$

Case C occurs with probability $p_C = \lambda\mu^2/(\lambda+\mu)^3$. Taking partial derivatives yields the joint PDF

$$f_C(t_1,t_2,t_3) = \begin{cases} \mu\left(2\lambda\mu e^{-\lambda t_2-\mu t_2-\mu t_3}+\lambda^2 e^{-\lambda t_2-\mu t_2-\mu t_3}+\mu^2 e^{-\lambda t_2-\mu t_2-\mu t_3}-\right. \\ \quad \lambda^2 e^{-\lambda t_2-\mu t_2-\lambda t_1-\mu t_1-\mu t_3}-2\mu\lambda e^{-\lambda t_2-\mu t_2-\lambda t_1-\mu t_1-\mu t_3}- \\ \quad \left.\mu^2 e^{-\lambda t_2-\mu t_2-\lambda t_1-\mu t_1-\mu t_3}\right) & t_1<t_2 \\ \mu\left(-\lambda^2 e^{-\lambda t_2-\mu t_2-\lambda t_1-\mu t_1-\mu t_3}-2\lambda\mu e^{-\lambda t_2-\mu t_2-\lambda t_1-\mu t_1-\mu t_3}+\right. \\ \quad \lambda^2 e^{-\lambda t_1-\mu t_1-\mu t_3}+2\lambda\mu e^{-\lambda t_1-\mu t_1-\mu t_3}+\mu^2 e^{-\lambda t_1-\mu t_1-\mu t_3}- \\ \quad \left.\mu^2 e^{-\lambda t_2-\mu t_2-\lambda t_1-\mu t_1-\mu t_3}\right) & t_2<t_1. \end{cases}$$

Case D. In case D, pairwise dependence occurs between customers 1 and 2 as well as between customers 2 and 3. This dependence leads to a more complicated version of the joint CDF occurring in five separate subcases based on the length of relative sojourn times. The joint CDF of T_1, T_2, and T_3 is given by the integral over region K, where $K = \{(y,z) : 0<y<t_1, y+z<t_2, 0<z<t_3\}$. Since the geometric form of the region over which y and z can vary depend on the relative sizes of t_1, t_2, and t_3, we must determine appropriate limits of integration for y and z separately for each of five possible cases:

1. $t_2 < \min\{t_1, t_3\}$

2. $t_1 < t_2 < t_3$

3. $t_3 < t_2 < t_1$

4. $\max\{t_1, t_3\} < t_2 < t_1 + t_3$

5. $t_2 > t_1 + t_3$.

The path segment distributions for T_1, T_2, T_3 are

$$\begin{aligned}(T_1, T_2, T_3) \quad \sim \quad & (\text{exponential}(\lambda + \mu) + Y, Y + \text{exponential}(\lambda + \mu) + Z, \\ & Z + \text{exponential}(\mu)),\end{aligned}$$

where $Y \sim \text{exponential}(\lambda + \mu)$ and $Z \sim \text{exponential}(\mu)$. The integrand for each case, denoted by I, is

$$I = \left(1 - e^{-(\lambda+\mu)(t_1-y)}\right)\left(1 - e^{-(\lambda+\mu)(t_2-y-z)}\right)\left(1 - e^{-\mu(t_3-z)}\right)(\lambda+\mu)e^{-y(\lambda+\mu)}\mu e^{-\mu z}.$$

Using this integrand, the joint CDF is

$$F_D(t_1, t_2, t_3) = \int\int_K I dy\, dz$$

where the region K is as formerly described. Case D occurs with probability $p_D = \lambda^2\mu/(\lambda+\mu)^3$. These five subcases correspond to the five shaded regions of integration sketched in Figures G.1 through G.5.

Subcase 1 of Case D. In this subcase, where $t_2 < \min\{t_1, t_3\}$, the region of integration occurs in the y, z plane as shown in Figure G.1, resulting in the following integral for the joint CDF:

$$F_{D_1}(t_1, t_2, t_3) = \int_0^{t_2}\int_0^{t_2-y} I\, dz\, dy.$$

Taking partial derivatives of the joint CDF yields the joint PDF

$$f_{D_1}(t_1,t_2,t_3) = \big(\lambda + \lambda^2 t_2 e^{t_2(\lambda+\mu)} + 2\lambda\mu t_2 e^{t_2(\lambda+\mu)} + \mu^2 t_2 e^{t_2(\lambda+\mu)} - \lambda e^{t_2(\lambda+\mu)} - \mu e^{t_2(\lambda+\mu)} + \mu\big)\, e^{-\lambda t_2 - \mu t_2 - \lambda t_1 - \mu t_1 - \mu t_3}\mu^2 \qquad t_2 < \min\{t_1,t_3\}.$$

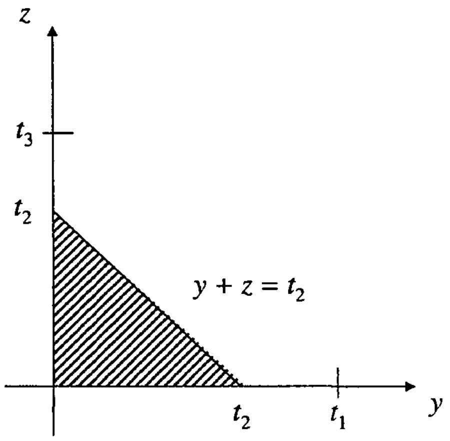


Figure G.1: Geometric form of subcase 1, case D, where $t_2 < \min\{t_1, t_3\}$.

Subcase 2 of Case D. In this subcase, where $t_1 < t_2 < t_3$, the region of integration occurs in the y, z plane as shown in Figure G.2, resulting in the following integral for the joint CDF:

$$F_{D_2}(t_1,t_2,t_3) = \int_0^{t_1}\int_0^{t_2-y} I dz\, dy.$$

At this point in the derivation, the joint PDFs become too cumbersome to express here. However, these expressions are tenable and are used to include all cases and achieve the trivariate distribution.

Subcase 3 of Case D. Subcase 3 is analogous to subcase 2 except that the ordering of sojourn time lengths changes to $t_3 < t_2 < t_1$. The region of integration again occurs in the y, z plane as shown in Figure G.3. This

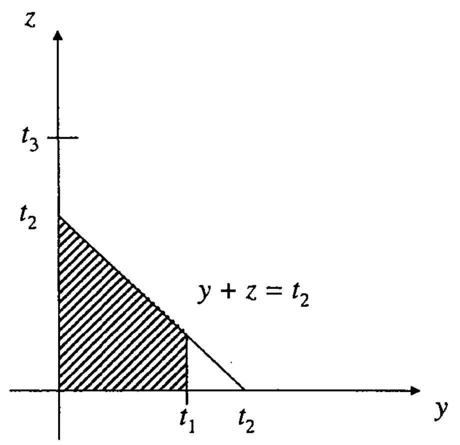


Figure G.2: Geometric form of subcase 2, case D, where $t_1 < t_2 < t_3$.

region results in the following expression for the joint CDF:

$$F_{D_3}(t_1, t_2, t_3) = \int_0^{t_3} \int_0^{t_2 - z} I \, dy \, dz.$$

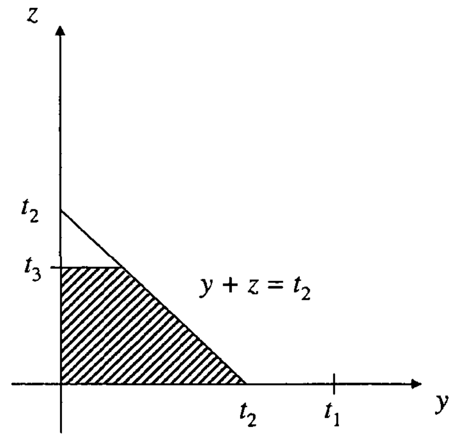


Figure G.3: Geometric form of subcase 3, case D, where $t_3 < t_2 < t_1$.

Subcase 4 of Case D. In this subcase, where $\max\{t_1, t_3\} < t_2 < t_1 + t_3$, the region of integration occurs in the y, z plane as shown in Figure G.4. Because of the shape of the region, the integration must be split into two parts. The joint CDF is found as

$$F_{D_4}(t_1,t_2,t_3) = \int_0^{t_2-t_3}\int_0^{t_3} I\,dz\,dy + \int_{t_2-t_3}^{t_1}\int_0^{t_2-y} I\,dz\,dy.$$

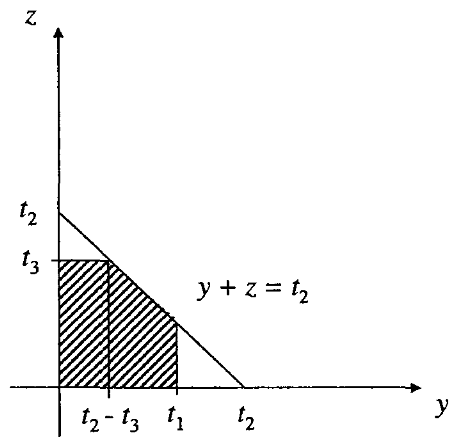


Figure G.4: Geometric form of subcase 4, case D, where $\max\{t_1,t_3\} < t_2 < t_1 + t_3$.

Subcase 5 of Case D. In this subcase, where $t_2 > t_1 + t_3$, the region of integration occurs in the y,z plane as shown in Figure G.5. The joint CDF is found as

$$F_{D_5}(t_1,t_2,t_3) = \int_0^{t_1}\int_0^{t_3} I\,dz\,dy.$$

An intermediate check of the validity of the derivations for case D can be accomplished by integrating each subcase joint PDF $f_{D_i}(t_1,t_2,t_3)$ over its corresponding support, which we will denote by $R_i, i = 1,2,\ldots,5$. Since case D results in a valid joint PDF when not weighted by p_D, the sum of the five subcases integrated appro-

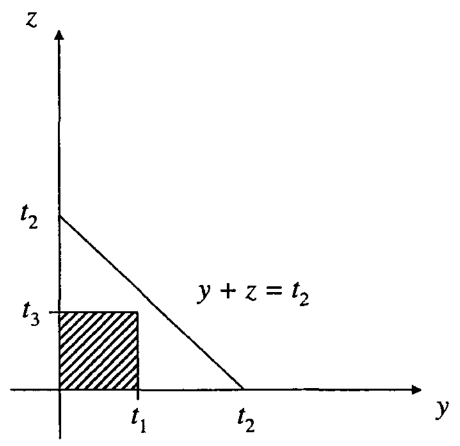


Figure G.5: Geometric form of subcase 5, case D, where $t_2 > t_1 + t_3$.

priately should be one. Thus it should be true that

$$\sum_{i=1}^{5}\left(\int\int_{R_i}\int f_{D_i}(t_1,t_2,t_3)dt_1\ dt_2\ dt_3\right) = 1.$$

Defining p_{D_i} as the contribution of subcase i to the above sum, the contributions of each of the five subcases follow.

For subcase 1, we must further consider the relative sizes of t_1 and t_3. There are two possibilities, $t_1 < t_3$ and $t_1 > t_3$, which lead to the contribution of subcase 1 as

$$\begin{aligned} p_{D_1} &= \int_0^\infty\int_0^{t_3}\int_0^{t_1} f_{D_1}(t_1,t_2,t_3)dt_2dt_1dt_3 + \int_0^\infty\int_0^{t_1}\int_0^{t_3} f_{D_1}(t_1,t_2,t_3)dt_2dt_3dt_1 \\ &= \frac{\mu\left(2\lambda\mu+\lambda^2+\mu^2\right)}{\left(\lambda+2\mu\right)^2\left(2\lambda+3\mu\right)}. \end{aligned}$$

Using the subcase 2 joint PDF, the contribution of the subcase is

$$p_{D_2} = \int_0^\infty\int_0^{t_3}\int_0^{t_2} f_{D_2}(t_1,t_2,t_3)dt_1dt_2dt_3 = \frac{2\lambda^3+3\lambda^2\mu+3\lambda\mu^2+\mu^3}{2\lambda^3+11\lambda^2\mu+20\lambda\mu^2+12\mu^3}.$$

Using the subcase 3 joint PDF, the contribution of the subcase is

$$p_{D_3} = \int_0^\infty \int_0^{t_1} \int_0^{t_2} f_{D_3}(t_1, t_2, t_3) dt_3 dt_2 dt_1 = \frac{\mu^2 (3\lambda + 4\mu)}{2(2\lambda + 3\mu)(2\mu + \lambda)^2}.$$

Using the subcase 4 joint PDF, the contribution of the subcase is

$$\begin{aligned} p_{D_4} &= \int_0^\infty \int_0^{t_3} \int_{t_3}^{t_1+t_3} f_{D_4}(t_1, t_2, t_3) dt_2 dt_1 dt_3 + \int_0^\infty \int_0^{t_1} \int_{t_1}^{t_1+t_3} f_{D_4}(t_1, t_2, t_3) dt_2 dt_3 dt_1 \\ &= \frac{\mu (3\lambda^2 + 7\lambda\mu + 4\mu^2)}{(2\lambda + 3\mu)(2\mu + \lambda)^2} \end{aligned}$$

Using the subcase 5 joint PDF, the contribution of the subcase is

$$p_{D_5} = \int_0^\infty \int_0^\infty \int_{t_1+t_3}^\infty f_{D_5}(t_1, t_2, t_3) dt_2 dt_1 dt_3 = \frac{\mu}{2(\lambda + 2\mu)}.$$

It can be shown by elementary algebra that, as desired,

$$\sum_{i=1}^{5} p_{D_i} = 1.$$

Case E. In case E, all three arrivals occur prior to the first departure, resulting in pairwise dependence between customers 1 and 2, and customers 2 and 3, as well as three-way dependence between the customers. This is the most complicated of the five cases, though it does share some commonality with case D. The joint CDF of T_1, T_2, and T_3 is given by the integral over region K, where $K = \{(y, z, w) : 0 < y, 0 < z, 0 < w, y + z < t_1, y + z + w < t_2, z + w < t_3\}$. Since the limits of integration for y, z, and w depend on the relative sizes of t_1, t_2, and t_3, we must determine appropriate limits separately for each of the following five possible cases:

1. $t_2 < \min\{t_1, t_3\}$

2. $t_1 < t_2 < t_3$

3. $t_3 < t_2 < t_1$

4. $t_1 < t_3 < t_2$ with $t_2 < t_1 + t_3$

5. $t_3 < t_1 < t_2$ with $t_2 < t_1 + t_3$.

The path segment distributions for T_1, T_2, T_3 are exponential$(\lambda+\mu)+Y+Z$, $Y+Z+W$, $Z+W+$ exponential(μ), where $Y \sim$ exponential$(\lambda+\mu)$, $Z \sim$ exponential(μ), and $W \sim$ exponential(μ). The integrand for each case, denoted by I, is

$$I = \left(1 - e^{-(\lambda+\mu)(t_1-y-z)}\right)\left(1 - e^{-\mu(t_3-z-w)}\right)(\lambda+\mu)e^{-y(\lambda+\mu)}\mu e^{-\mu z}\mu e^{-\mu w}.$$

Using this integrand, the joint CDF is

$$F_E(t_1, t_2, t_3) = \iiint_K I\, dy\, dz\, dw,$$

where the region K is as formerly described. Case E occurs with probability $p_E = \lambda^2/(\lambda+\mu)^2$.

For each of the five subcases of case E, the integrals required to compute the joint CDF are provided. However, the three-dimensional figures associated with these subcases are not included.

Subcase 1 of Case E. The joint CDF is found as

$$F_{E_1}(t_1, t_2, t_3) = \int_0^{t_2}\int_0^{t_2-y}\int_0^{t_2-y-z} I\, dw\, dz\, dy.$$

Subcase 2 of Case E. The joint CDF is found as

$$F_{E_2}(t_1, t_2, t_3) = \int_0^{t_1}\int_0^{t_1-y}\int_0^{t_2-y-z} I\, dw\, dz\, dy.$$

Subcase 3 of Case E. The joint CDF is found as

$$F_{E_3}(t_1, t_2, t_3) = \int_0^{t_3}\int_0^{t_3-z}\int_0^{t_2-w-z} I\, dy\, dw\, dz.$$

Subcase 4 of Case E. The joint CDF is found as

$$F_{E_4}(t_1,t_2,t_3) = \int_0^{t_2-t_3}\int_0^{t_1-y}\int_0^{t_3-z} I\,dw\;dz\;dy + \int_{t_2-t_3}^{t_1}\int_0^{t_1-y}\int_0^{t_2-y-z} I\,dw\;dz\;dy.$$

Subcase 5 of Case E. The joint CDF is found as

$$F_{E_5}(t_1,t_2,t_3) = \int_0^{t_2-t_1}\int_0^{t_3-w}\int_0^{t_1-z} I dy\;dz\;dw + \int_{t_2-t_1}^{t_3}\int_0^{t_3-w}\int_0^{t_2-w-z} I dy\;dz\;dw.$$

After computing the joint CDFs for each subcase and taking partial derivatives to find the joint PDFs, the overall joint PDF for the sojourn times of the first three customers can be computed as the mixture

$$f_{T_1,T_2,T_3}(t_1,t_2,t_3) = p_A \cdot f_A + p_B \cdot f_B + p_C \cdot f_C + p_D \cdot f_D + p_E \cdot f_E.$$

If the joint CDFs calculated for the five cases, A through E, are correct, then the resulting joint PDFs (produced by taking the third order partial derivatives of the joint CDFs with respect to t_1, t_2, and t_3) should each yield the result 1 when integrated over the first octant of (t_1, t_2, t_3) space. In order to perform this consistency check for the CDF in case E, we must partition the first octant into regions, each of which has a different algebraic expression for the PDF, and determine appropriate limits of integration for each region. There are six such regions, five corresponding to the five subcases listed for case E above, and one region on which the PDF is 0, given by the impossible situation $t_2 > t_1 + t_3$ (this inequality is never satisfied since the time interval of duration t_2 is a subset of the union of the time intervals of duration t_1 and t_3). These six regions in the first octant are bordered by the coordinate planes and by the following three planes, all of which pass through $(0,0,0)$: $t_1 = t_2$, $t_1 = t_3$, $t_2 = t_3$, and $t_2 = t_1 + t_3$. Figure G.6 depicts these six regions by showing their intersection with the plane $t_1 + t_2 + t_3 = 2$. Each region "begins" at $(0,0,0)$, has three planar sides and is of infinite extent. To check the

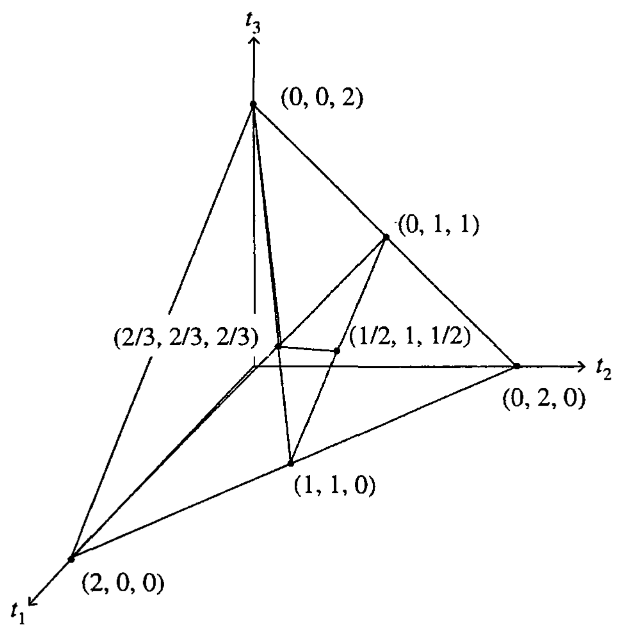


Figure G.6: Geometric presentation of the five subcases for case E.

accuracy of the trivariate distribution, $E[T_1]$, $E[T_2]$, $E[T_3]$, Cov(T_1, T_2), Cov(T_1, T_3), and Cov(T_2, T_3) were computed. The results matched exactly those produced by the `Queue(X, Y, n, k, s)` procedure and were further supported by Monte Carlo simulation. Additionally, the marginal distributions of the sojourn times of the first three customers were computed from the trivariate distribution, and these marginal distributions matched those computed by conditioning. The Maple code to calculate all joint PDFs is given at `www.math.wm.edu/leemis/trivariate.txt`.

Appendix H

Three Customer Next-Event Simulation for Computing Covariance

The following simulation is written for R/S-plus providing the verification for the covariance calculations in Section 6.5.2 for $n = 3$ customers.

```
# --------------------------------------------------
# Discrete-event simulation for customers 1, 2, and
# 3 to calculate their covariance.
# Name            : simt1t2t3.txt
# Author          : Billy Kaczynski
# Language        : R/S-Plus
# Latest Revision : 11/04/08
# --------------------------------------------------

N <- 100000
t1 <- rexp(N, 2)
a2 <- rexp(N, 1)
a3 <- a2 + rexp(N, 1)
c2 <- c(1:N)
c3 <- c(1:N)
for (i in 1:N) {
  if (t1[i] > a2[i]) c2[i] <- t1[i] + rexp(1, 2)
  else c2[i] <- a2[i] + rexp(1, 2)
}
t2 <- c2 - a2
for (i in 1:N) {
  if (a3[i] > c2[i]) {c3[i] <- a3[i] + rexp(1, 2)}
  if (a3[i] < t1[i]) {c3[i] <- c2[i] + rexp(1, 2)}
  if (a3[i] < c2[i] & a3[i] > t1[i]) c3[i] <- c2[i] + rexp(1, 2)
}
t3 <- c3 - a3

covt1t2 <- mean(t1 * t2) - mean(t1) * mean(t2)
covt1t2
covt1t3 <- mean(t1 * t3) - mean(t1) * mean(t3)
covt1t3
covt2t3 <- mean(t2 * t3) - mean(t2) * mean(t3)
covt2t3
```

Appendix I

Paths for $n = 3$ Customers in an $M/M/1$ Queue

The number of arrival/departure paths possible for $n = 3$ customers is the third Catalan number, or

$$C_3 = \frac{(2n)!}{n!(n+1)!} = 5.$$

Figure I.1 depicts the five cases as paths from the bottom left node to the top right node of each figure. The cases are ordered according to the rows of C, the case matrix, which are created by the prefix-shift algorithm presented in Section 6.6. In the case matrix C, an arrival is annotated by a 1 and a departure by a -1. This algorithm guarantees the generation of all such possible paths. For $n = 3$, the case matrix C is

$$C = \begin{bmatrix} 1 & -1 & 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & -1 & -1 & -1 \end{bmatrix}.$$

Moving right in the figure denotes an arrival and moving up denotes a service completion. Diagonal moves are not permitted. These paths provide the methodology for calculating the probability associated with each case along with the appropriate probability distribution for each case.

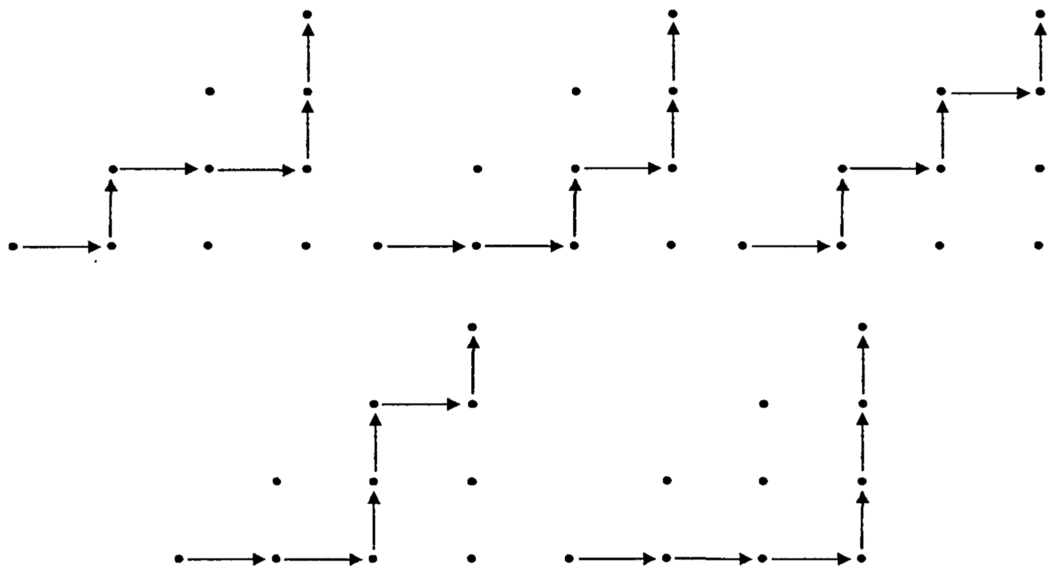


Figure I.1: Five paths for $n = 3$ customers' sojourn times in an $M/M/1$ queue.

Appendix J

Exact Covariance Calculations in an $M/M/1$ Queue

The list of procedures presented here collectively calculates the covariance between two specific customers in an $M/M/1$ queue, without regard to the usual traffic intensity requirement $\rho < 1$. Select procedures also return interesting pieces of information, such as the joint PDF for customers a and b, the probability a specific case occurs, a vector of probabilities for all cases, etc. Each procedure is summarized in a comment block and required arguments are provided.

J.1 cases(n)

```
#cases(n)
#----------------------------------------------------------
# Generates all possible arrival/departure sequences for
# n customers in an M/M/1 queue initially empty and idle.
# Resulting list of sequences consists of 1's and -1's,
# where a 1 is an arrival and a -1 is a departure. The
# sequences are returned in the matrix C, of dimension c
# by 2n, where c in the nth Catalan number calculated as
# (2n)! / n! / (n + 1)!.  The procedure calls ini(n) to
# initialize the first sequence in the matrix, then uses
# the procedures swapa(n, A), swapb(n, A), and okay(n, A)
# to create the remaining sequences according to a
# prefix shift algorithm. For each row in the resulting
# matrix, an associated path matrix can be generated via
# the procedure path(n, A).
#
# Name            : cases.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/12/09
#---------------------------------------------------------

cases := proc(n)
   options remember;
   local c, C, i;
   c := (2 * n)! / n! / (n + 1)!;
   C := Matrix(c, 2 * n);
   for i from 1 to c do
     if (i = 1) then
       C[[i], 1 .. -1] := ini(n);
     else
       C[[i], 1 .. -1] := swapa(n, C[[i - 1], 1 .. -1]);
     fi:
     if (okay(n, C[[i], 1 .. -1]) = 0) then
       C[[i], 1 .. -1] := swapb(n, C[[i - 1], 1 .. -1]);
     fi:
   od:
   return C;
end:
```

J.2 ini(n)

```
#ini(n)
#---------------------------------------------------------
# Initializes the matrix C according the Ruskey and
# Williams (2008). Returns the first row of C to enable
# use of their prefix shift algorithm. Requires the
# parameter n, the number of customers.
#
# Name            : ini.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/11/09
# --------------------------------------------------------

ini := proc(n)
   options remember;
   local L, i;
   L := Matrix(1, 2 * n, -1):
   L[1, 1] := 1:
   for i from 3 to (n + 1) do
     L[1, i] := 1
   od:
   for i from (n + 2) to (2 * n) do
     L[1, i] := -1
   od:
   return L:
 end:
```

J.3 swapa(n, A)

```
# swapa(n, A)
#-------------------------------------------------------
# Conducts the (k + 1)st prefix shift in creating all
# instances of the case matrix, C, according to Ruskey
# and Williams (2008).  Requires the arguments n, number
# of customers, and A, row i of C.  Returns the
# successor of C[i, ] to be checked by the procedure
# okay(n, A).
#
# Name            : swapa.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/11/09
# -------------------------------------------------------

swapa := proc(n, A)
   local check, i, temp1, j, R;
   R := A;
   check := 1;
     for i from 2 to (2 * n - 1) do
       if ((R[1, i] = -1) and (R[1, (i + 1)] = 1)) then
         temp1 := R[1, i + 2];
         R[1 .. 1, 3 .. (i+2)] := R[1 .. 1, 2 .. (i + 1)];
         check := 0;
         R[1, 2] := temp1;
       fi:
       if (check = 0) then break fi;
     od:
   return R:
end:
```

J.4 swapb(n, A)

```
# swapb(n, A)
#--------------------------------------------------------
# Conducts the (k)th prefix shift in creating all
# instances of the case matrix, C, according to Ruskey
# and Williams (2008).  Requires the arguments n, number
# of customers, and A, row i of C.  Returns the
# successor of C[i, ] in the event that okay(n, a)
# identifies the output of swapa(n, a) invalid.
#
# Name            : swapb.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/11/09
# -------------------------------------------------------

swapb := proc(n, B)
  local check, i, temp, j;
  check := 1;
  for i from 2 to (2 * n - 2) do
    if ((B[1, i] = -1) and (B[1, (i + 1)] = 1)) then
      temp := B[1, i + 1];
      B[[1 .. 1], [3 .. (i + 1)]] := B[1 .. 1, 2 .. (i)];
      check := 0;
      B[1, 2] := temp;
    fi:
    if (check = 0) then break fi;
  od:
  return B:
end:
```

J.5 okay(n, E)

```
# okay(n, E)
#---------------------------------------------------------
# Checks the output of swapa(n, A) for an illegal prefix
# shift, meaning the result contains an impossible
# arrival/service sequence.  Requires arguments n,
# number of customers, and E, resulting vector from
# swapa(n, A).  If the (k + 1)st shift is legal,
# okay(n, E) returns 1, signifying the correct successor
# in C.  If the (k + 1)st shift is illegal, okay(n, E)
# returns 0 which in turn calls swapb(n, A) to produce
# the correct successor row in C.
#
# Name            : okay.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/11/09
# ---------------------------------------------------------

okay := proc(n, E)
   local s, i, test;
   test := 1:
   s := 0;
   for i from 1 to (2 * n - 1) do
     s := s + E[1, i];
     if (s < 0) then
       test := 0;
       break;
     fi:
   od:
   return test;
 end:
```

J.6 path(n, A)

```
# path(n, A)
#-----------------------------------------------------------
# Creates a path matrix of size (n + 1) by (n + 1), where
# 1's represent the arrival/service sequence for a given
# row of the case matrix C.  All other elements in the
# path matrix are 0.  The path starts at the lower-left
# corner of the matrix and moves to the upper-right
# corner.  The first leg of the path is always the
# arrival of customer 1 represented by the entries in the
# [n + 1, 1] and [n + 1, 2] positions.  A 1 to the right
# of the previous 1 signifies an arrival, while a 1 above
# the previous 1 signifies a service completion.  The
# procedure requires the arguments n, number of customers
# and A, a row from the case matrix C.
#
# Name            : path.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/12/09
# ----------------------------------------------------------

path := proc(n, A)
   local j, row, col, pat;
   row := n + 1;
   col := 2;
   pat := Matrix(n + 1, n + 1);
   pat[n + 1, 1] := 1;
   pat[n + 1, 2] := 1;
   for j from 2 to (2 * n) do
     if (A[1, j] = 1) then
       col := col + 1;
       pat[row, col] := 1;
     else
       row := row - 1;
       pat[row, col] := 1;
     fi;
   od:
   return pat;
 end:
```

J.7 Cprime(n, C)

```
# Cprime(n, C)
#-----------------------------------------------------------
# Produces the matrix defined as C', that is the
# distribution segment matrix where each row represents
# the distribution segments for the case represented by
# the corresponding row in the case matrix C.  The
# elements of C' are limited to a 0, 1, and 2, where 0
# implies no sojourn time distribution segment due to an
# emptying of the system, 1 implies a competing risk of an
# arrival or completion of service and is distributed
# exponential(lambda + mu), and a 2 implies a service
# completion distribution leg which is distributed
# exponential(mu).  The matrix C' has the same number of
# rows as C, and 2n - 1 columns.  Cprime(n, C) calls
# path(n, A) and uses the path matrix to determine the
# appropriate probability distribution function segments.
# The procedure requires the arguments n, number of
# customers and C, the case matrix.
#
# Name            : Cprime.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/12/09
# -----------------------------------------------------------

Cprime := proc(n, C)
   local prime, i, pat, dist, j, row, col;
   prime := Matrix(RowDimension(C), 2 * n - 1);
   for i from 1 to RowDimension(C) do
     row := n + 1;
     col := 2;
     pat := path(n, C[[i], 1 .. -1]);
     dist := Matrix(1, 2 * n - 1);
     for j from 1 to (2 * n - 1) do
       if (pat[row - 1, col] = 1) and (col < n + 1) then
         row := row - 1;
         dist[1, j] := 1;
       elif (pat[row - 1, col] = 1) and (col = n + 1) then
         row := row - 1;
```

```
        dist[1, j] := 2;
      elif (pat[row, col + 1] = 1) and (row + col > n + 2) then
        col := col + 1;
        dist[1, j] := 1;
      else
        col := col + 1;
        dist[1, j] := 0;
      fi;
    od;
    prime[[i], 1 .. -1] := dist;
  od;
  return prime;
end:
```

J.8 caseprob(n, P)

```
# caseprob(n, P)
#-----------------------------------------------------------------
# Computes the probability associated with a given row of the
# case matrix C as represented by the path created by
# path(n, A).  Similar to how C' identifies the appropriate
# distribution segments along the path, caseprob(n, P)
# identifies the appropriate probability for each leg of the
# path based on whether a competing risk occurs.  Requires the
# arguments n, number of customers and P, the path of a given
# case.  Returns the probability of the case passed to the
# procedure.
#
# Name            : caseprob.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/12/09
# ----------------------------------------------------------------

caseprob := proc(n, P)
   global X, Y;
   local p, j, row, col;
   p := 1;
   row := n + 1;
   col := 2;
   for j from 1 to (2 * n - 1) do
     if (P[row - 1, col] = 1) and (col < n + 1) then
       row := row - 1;
       p := p * 1 / Mean(Y) / (1 / Mean(X) + 1 / Mean(Y));
     elif (P[row - 1, col] = 1) and (col = n + 1) then
       row := row - 1;
     elif (P[row, col + 1] = 1) and (row + col > n + 2) then
       col := col + 1;
       p := p * 1 / Mean(X) / (1 / Mean(X) + 1 / Mean(Y));
     else col := col + 1;
     fi;
   od:
   return p;
end:
```

J.9 probvec(n)

```
# probvec(n)
#------------------------------------------------------------
# Uses the procedure caseprob(n, P) successively to build a
# vector of probabilities, one for each case of the C matrix.
# This vector has length (2n)! / n! / (n = 1)!, which the the
# n-th Catalan number.  Requires the argument n, the number
# of customers.
#
# Name            : probvec.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/12/09
# ------------------------------------------------------------

probvec := proc(n)
   local i, p, c;
   c := (2 * n)! / n! / (n + 1)!;
   p := Vector(c);
   for i from 1 to c do
     p[i] := caseprob(n, path(n, cases(n)[[i], 1  ..  -1]));
   od:
   return p;
 end:
```

J.10 Tmat(a, b, A)

```
# Tmat(a, b, A)
#--------------------------------------------------------------
# Creates the 2 by 2 matrix for determining whether selected
# customer sojourn times are independent.  Also provides
# information on the required distribution segments for
# calculating the joint distribution between two customers.
# Requires the arguments a, the first customer of interest in
# the system, b, the second customer of interest in the
# system, and A, a single row of the case matrix C,
# representing a given case.  It uses this row of C to
# identify the start and finish indices for customers a and
# b.  If these indices overlap, sojourn times are dependent,
# if they do not overlap the sojourn times are independent.
#
# Name            : Tmat.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/12/09
# --------------------------------------------------------------

Tmat := proc(a, b, A)
  local sta, fina, stb, finb, indexa, indexb, i, T;
  indexa := 0;
  indexb := 0;
  for i from 1 to ColumnDimension(A) do
    if A[1, i] = 1 then
      indexa := indexa + 1;
      if indexa = a then sta := i
      fi:
      if indexa = b then stb := i
      fi:
    elif A[1, i] = -1 then
      indexb := indexb - 1;
      if indexb = -(a) then fina := i
      fi:
      if indexb = -(b) then finb := i
      fi:
    fi:
  od:
```

```
  T := Matrix(2, 2, [[sta, fina], [stb, finb]]);
  return T;
end:
```

J.11 inde(a, b, T, A)

```
# inde(a, b, T, A)
#--------------------------------------------------------------------
# Calculates the case-specific joint cumulative distribution
# function for customers a and b whose sojourn times are
# independent by multiplying the CDFs of each customer.  The
# individual customer CDFs are calculated by determining the
# type and number of distribution legs using the arguments a, the
# first customer of interest, b, the second customer of interest,
# T, the resulting matrix from the call Tmat(a, b, A), and A, the
# row of C' associated with the specific case.  The CDF forms
# for each case arise from appropriately defined random variables
# in APPL. The procedure returns the joint cumulative distribution
# function in a vector of length two, where both elements are
# identical in order to match the piecewise result for customers
# with dependent sojourn times.
#
# Name            : inde.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/12/09
# --------------------------------------------------------------------

inde := proc(a, b, T, A)
  options remember;
  global X, Y;
  local i, dist1, dist2, jcdf, expa, expb;
  expa := 0;
  expb := 0;
  for i from 1 to (T[1, 2] - T[1, 1]) do
    if A[1, T[1, 1] + i - 1] = 1 then
      expa := expa + 1;
    elif A[1, T[1, 1] + i - 1] = 2 then
      expb := expb + 1;
    fi:
  od:
  if expb = 0 then dist1 := ErlangRV(1 / Mean(X) + 1 / Mean(Y), expa)
  elif expa = 0 then dist1 := ErlangRV(1 / Mean(Y), expb)
  else dist1 := [[unapply(conv(expa, expb), t)], [0, infinity],
              ["Continuous", "PDF"]];
  fi:
  expa := 0;
  expb := 0;
  for i from 1 to (T[2, 2] - T[2, 1]) do
    if A[1, T[2, 1] + i - 1] = 1 then
      expa := expa + 1;
    elif A[1, T[2, 1] + i - 1] = 2 then
      expb := expb + 1;
    fi:
  od:
  if expb = 0 then dist2 := ErlangRV(1 / Mean(X) + 1 / Mean(Y), expa)
  elif expa = 0 then dist2 := ErlangRV(1 / Mean(Y), expb)
  else dist2 := [[unapply(conv(expa, expb), t)], [0, infinity],
                ["Continuous", "PDF"]];
  fi:
  jcdf := apply(op(CDF(dist1)[1]), t[a]) *
          apply(op(CDF(dist2)[1]), t[b]);
  return Matrix([jcdf, jcdf]);
end:
```

J.12 dep(a, b, T, A)

```
# dep(a, b, T, A)
#-----------------------------------------------------------------
# Calculates the case-specific joint cumulative distribution
# function for customers a and b whose sojourn times are dependent
# by conditioning on the overlap distribution segment(s).  The
# customer sojourn time segments are divided up into their
# associated independent and dependent (overlap) portions.  This
# amounts to three segments, customer a's independent portion
# defined as dist1, customer b's independent portion defined as
# dist2, and the dependent overlap portion defined as dist3.  The
# joint cumulative distribution function has two pieces, for the
# cases when t[a] < t[b] and t[b] < t[a].
#
# Name            : dep.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/12/09
# ----------------------------------------------------------------

dep := proc(a, b, T, A)
  options remember;
  global X, Y;
  local i, expa, expb, dist1, dist2, dist3, jcdftop, jcdfbot;
  expa := 0;
  expb := 0;
  for i from 1 to (T[2, 1] - T[1, 1]) do
    if A[1, T[1, 1] + i - 1] = 1 then
      expa := expa + 1;
    elif A[1, T[1, 1] + i - 1] = 2 then
      expb := expb + 1;
    fi:
  od:
  if expb = 0 then dist1 := ErlangRV(1 / Mean(X) + 1 / Mean(Y), expa)
  elif expa = 0 then dist1 := ErlangRV(1 / Mean(Y), expb)
  else dist1 := [[unapply(conv(expa, expb), t)], [0, infinity],
                ["Continuous", "PDF"]];
  fi:
  expa := 0;
  expb := 0;
  for i from 1 to (T[2, 2] - T[1, 2]) do
    if A[1, T[1, 2] + i - 1] = 1 then
      expa := expa + 1;
    elif A[1, T[1, 2] + i - 1] = 2 then
      expb := expb + 1;
    fi:
  od:
  if expb = 0 then dist2 := ErlangRV(1 / Mean(X) + 1 / Mean(Y), expa)
  elif expa = 0 then dist2 := ErlangRV(1 / Mean(Y), expb)
  else dist2 := [[unapply(conv(expa, expb), t)], [0, infinity],
                ["Continuous", "PDF"]];
  fi:
  expa := 0;
  expb := 0;
  for i from 1 to (T[1, 2] - T[2, 1]) do
    if A[1, T[2, 1] + i - 1] = 1 then
      expa := expa + 1;
    elif A[1, T[2, 1] + i - 1] = 2 then
      expb := expb + 1;
    fi:
  od:
  if expb = 0 then dist3 := ErlangRV(1 / Mean(X) + 1 / Mean(Y), expa)
  elif expa = 0 then dist3 := ErlangRV(1 / Mean(Y), expb)
  else dist3 := [[unapply(conv(expa, expb), t)], [0, infinity],
                ["Continuous", "PDF"]];
  fi:
  jcdftop := int(apply(op(CDF(dist1)[1]), t[a] - y) *
             apply(op(CDF(dist2)[1]), t[b] - y) *
             apply(op(PDF(dist3)[1]), y), y = 0 .. t[a]);
  jcdfbot := int(apply(op(CDF(dist1)[1]), t[a] - y) *
             apply(op(CDF(dist2)[1]), t[b] - y) *
             apply(op(PDF(dist3)[1]), y), y = 0 .. t[b]);
  return Matrix([jcdftop, jcdfbot]);
end:
```

J.13 jpdf(a, b, n)

```
# jpdf(a, b, n)
#-----------------------------------------------------------------
# Creates the case joint cumulative distribution functions in a
# matrix with dimension (2n)! / n! / (n + 1)! by 2.  Calls the
# procedure Tmat(a, b, A) and depending on the structure
# returned, calls the procedures inde(a, b, T, A) or
# dep(a, b, T, A) to generate the appropriate case-wise joint
# cumulative distribution function.  Requires arguments a, the
# index of the first customer of interest, b, the index of the
# second customer of interest, and n, the number of customers.
#
# Name            : jpdf.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/13/09
# -----------------------------------------------------------------

jpdf := proc(a, b, n)
  local C, i, c, dist, T;
  C := cases(n);
  c := (2 * n)! / n! / (n + 1)!;
  dist := Matrix(c, 2);
  for i from 1 to c do
    T := Tmat(a, b, C[[i], 1 .. -1]);
    if T[1, 2] < T[2, 1] then
      dist[[i], 1 .. -1] := inde(a, b, T, Cprime(n, C[[i], 1 .. -1]));
    else
      dist[[i], 1 .. -1] := dep(a, b, T, Cprime(n, C[[i], 1 .. -1]));
    fi:
  od:
  return dist;
end:
```

J.14 conv(m, n)

```
# conv(m, n)
#-------------------------------------------------------------------
# Sums the appropriate distribution segments for the independent
# and dependent portions of customer sojourn times bypassing the
# required calls to Convolution(X, Y) in APPL by rewriting the
# integral as sums.  Saves significant CPU time by recognizing
# that these segments can all be written as a sum of Erlang random
# variables.  Requires the arguments m, the number of
# expon(lambda + mu) segments and n, the number of expon(mu)
# segments.
#
# Name            : conv.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 02/2/09
# -------------------------------------------------------------------

conv := proc(m, n)
  options remember;
  global X, Y;
  local f1, f2, r, x, t, pdf;
  f1 := unapply((-1) ^ r * (m - 1 + x)! * (t) ^ (m - 1 + x - r) /
        (m - 1 + x - r)! / (1 / Mean(Y) - (1 / Mean(X) +
        1 / Mean(Y))) ^ (r + 1), r, x);
  f2 := unapply((-1) ^ x * (n - 1)! / (n - 1 - x)! / x! *
        w ^ (n - 1 - x) * exp((1 / Mean(Y) - (1 / Mean(X) +
        1 / Mean(Y))) * t), x);
  pdf := (1 / Mean(X) + 1 / Mean(Y)) ^ m * (1 / Mean(Y)) ^ n *
         exp(-(1 / Mean(Y)) * w) / (m - 1)! / (n - 1)! * add(f2(x)
         * add(f1(r, x), r = 0 .. m - 1 + x), x = 0 .. n - 1);
  return simplify(subs(t = w, pdf) - subs(t = 0, pdf));
end:
```

J.15 Cov(a, b, n)

```
# Cov(X, Y, a, b)
#-----------------------------------------------------------------
# Mixes the results returned by probvec(n) and jpdf(a, b, n) to
# compute the joint cumulative distribution function encompassing
# all cases.  Differentiates the results to produce the piecewise
# joint probability distribution function.  Calls Queue(n, k, s)
# to find the appropriate expected values for the customers of
# interest, then uses the expected values along with the expected
# value E(T[a]T[b]) found using the joint probability distribution
# function to compute the covariance as Cov(T[a], T[b]) =
# E(T[a]T[b]) - E(T[a])E(T[b]).  Requires the arguments X, the
# distribution of time between arrivals in the APPL list-of-lists
# format, Y, the service time distribution in the list-of-lists
# format, a, the index of the first customer of interest, and b,
# the index of the second customer of interest (a < b).
#
# Name            : cov.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 01/13/09
# -----------------------------------------------------------------

Cov := proc(X, Y, a, b)
  global lambda1, lambda2;
  local JPDFMAT, PVEC, JPDF, fta, ftb, Etatb, Eta, Etb, Cov;
  JPDFMAT := jpdf(a, b, b);
  PVEC := probvec(b);
  JPDF := Transpose(JPDFMAT) . PVEC;
  fta := simplify(diff(diff(JPDF[1], t[a]), t[b]));
  ftb := simplify(diff(diff(JPDF[2], t[a]), t[b]));
  Etatb := int(int(t[a] * t[b] * fta, t[a] = 0 .. t[b]),
           t[b] = 0 .. infinity) + int(int(t[a] * t[b] *
           ftb, t[b] = 0 .. t[a]), t[a] = 0 .. infinity);
  Eta := Mean(Queue(X, Y, a, 0, 1));
  Etb := Mean(Queue(X, Y, b, 0, 1));
  Cov := Etatb - Eta * Etb;
  return Cov;
end:
```

Appendix K

Sojourn Time Monte Carlo Simulation

The following S-Plus/R code is a Monte Carlo simulation capturing sojourn times of the first n customers in an $M/M/1$ queue that is initially empty and idle and computing the variance of their sample mean. The simulation is used to verify the calculations shown in Example 5 in Chapter 6. The algorithm used to create the code is from Leemis and Park (2006).

```
# -----------------------------------------------------------
# Monte Carlo simulation for customers 1, 2, ... , n to
# calculate their sojourn times in an M/M/1 queue.
# Name             : tn.txt
# Author           : Billy Kaczynski
# Language         : R/S-Plus
# Latest Revision : 02/04/09
# -----------------------------------------------------------

lambda <- 1
mu <- 10 / 9
N <- 1000000
n <- 10
Tbar <- matrix(0, N)
for (j in 1:N) {
  C <- matrix(0, n)
  D <- matrix(0, n)
  A <- matrix(0, n)
  S <- rexp(n, mu)
  C[1] <- S[1]
  for (i in 2:n) {
    A[i] <- A[i - 1] + rexp(lambda)
  }
  i <- 1
  while (i < n) {
    i <- i + 1
    if (A[i] < C[i - 1]) {
      D[i] <- C[i - 1] - A[i]
    }
    else
      D[i] <- 0
      C[i] <- A[i] + D[i] + S[i]
  }
  Tbar[j] <- mean(D + S)
}
var(Tbar)
```

Appendix L

Counting Sequences with k Customers Present at Time Zero

It is natural to ask the following question: If a queueing system is preloaded with k customers at time zero, and n more customers arrive after time zero, how many sequences of arrival and service times are possible? We will develop a formula to answer that question.

Any particular sequence of arrivals and servicings can be represented by a vector of 1's and -1's as before, but since the system is preloaded with k customers, the vector must begin with k ones. For example, if a system is preloaded with $k = 2$ customers and $n = 1$ additional customer arrives later, the three possible sequences of arrivals and servicings are represented by the vectors

$$(1, 1, 1, -1, -1, -1), (1, 1, -1, 1, -1, -1), \text{ and } (1, 1, -1, -1, 1, -1).$$

Denote the number of ways that n additional customers can arrive and be served, given that the system is preloaded with k customers that must also be served by $C(n|k)$. Using this notation, $C(n|0) = C_n$, the nth Catalan number, as defined earlier.

Appendix L. Counting Sequences with k Customers Present at Time Zero

It is easy to see that $C(n|1) = C_{n+1}$ for $n = 1, 2, 3, \ldots$ since both can be determined by counting vectors with $n+1$ ones and $n+1$ minus ones, each vector beginning with a one.

Developing a general formula for $C(n|k)$ is based on two recursion formulas, one for k even and one for k odd. If $k \geq 2$ is even, the vectors of $k+n$ ones and $k+n$ minus ones that comprise $C(n|k)$ are the same as the vectors of $(n+1)+(k-1)$ ones and $(n+1)+(k-1)$ minus ones that comprise $C(n+1|k-1)$, except for those vectors of $C(n+1|k-1)$ that begin with $k-1$ ones followed immediately by a minus one, of which there are $C(n+1|k-2)$, because the $k-1$ ones followed by a minus one effectively constitute a preload of $k-2$ customers that must be followed by $n+1$ arrivals. Thus if $k \geq 2$ is even,

$$C(n|k) = C(n+1|k-1) - C(n+1|k-2)$$

for $n = 1, 2, \ldots$.

If $k \geq 3$ is odd, we again form the vectors of $C(n|k)$ by removing inappropriate vectors from those that comprise $C(n+1|k-1)$. However, in this case we remove vectors that begin with $k-1$ ones followed by a minus one in two steps. We first remove vectors that begin with $k-1$ ones followed by the ordered pair $(-1, 1)$, of which there are $C(n|k-1)$ because the $k-1$ ones followed by $(-1, 1)$ effectively constitute a preload of $k-1$ customers that must be followed by n arrivals. We then remove vectors that begin with $k-1$ ones followed by $(-1, -1)$, of which there are $C(n+1|k-3)$, because the $k-1$ ones followed by $(-1, -1)$ effectively constitute a preload of $k-3$ customers that must be followed by $n+1$ arrivals. Thus if $k \geq 3$ is odd,

$$C(n|k) = C(n+1|k-1) - C(n|k-1) - C(n+1|k-3)$$

for $n = 1, 2, \ldots$.

Using the fact that $C(n|0) = C_n$ and $C(n|1) = C_{n+1}$ for $n = 1, 2, \ldots$ and the above two recursion formulas, we find that for $n = 1, 2, \ldots,$

$$\begin{aligned}
C(n|2) &= C_{n+2} - C_{n+1}, \\
C(n|3) &= C_{n+3} - 2C_{n+2}, \\
C(n|4) &= C_{n+4} - 3C_{n+3} + C_{n+2}, \\
C(n|5) &= C_{n+5} - 4C_{n+4} + 3C_{n+3}, \\
C(n|6) &= C_{n+6} - 5C_{n+5} + 6C_{n+4} - C_{n+3}, \\
C(n|7) &= C_{n+7} - 6C_{n+6} + 10C_{n+5} - 4C_{n+4}.
\end{aligned}$$

The pattern emerging in these formulas can be expressed by the following general result:

$$C(n|k) = \sum_{j=0}^{\lfloor k/2 \rfloor} (-1)^j \binom{k-j}{j} C_{n+k-j}$$

for $k = 0, 1, 2, \ldots$ and $n = 1, 2, \ldots$, where $\lfloor \cdot \rfloor$ denotes the greatest integer function.

It is easy to see that this general result is true for $k = 0$ and $k = 1$. In order to show that it is true for $k \geq 2$, we proceed by induction, showing that if the result is true for $0, 1, 2, \ldots, k-1$, then the result is true for k. Since there are different recursion formulas for k even and k odd, we must treat these two cases separately.

First let us suppose that $k \geq 2$ is even and that the general result is true for $0, 1, 2, \ldots, k-1$. Then for $n = 1, 2, \ldots,$

$$\begin{aligned}
C(n|k) &= C(n+1|k-1) - C(n+1|k-2) \\
&= \sum_{j=0}^{\lfloor \frac{k-1}{2} \rfloor} (-1)^j \binom{k-1-j}{j} C_{n+k-j} - \sum_{j=0}^{\lfloor \frac{k-2}{2} \rfloor} (-1)^j \binom{k-2-j}{j} C_{n+k-1-j}.
\end{aligned}$$

Shifting the index of summation by one in the second sum, the previous line becomes

$$\begin{aligned}
C(n|k) &= \sum_{j=0}^{\frac{k-2}{2}}(-1)^j\binom{k-1-j}{j}C_{n+k-j}+\sum_{j=1}^{\frac{k}{2}}(-1)^j\binom{k-1-j}{j-1}C_{n+k-j} \\
&= C_{n+k}+\sum_{j=1}^{\frac{k-2}{2}}(-1)^j\left[\binom{k-1-j}{j}+\binom{k-1-j}{j-1}\right]C_{n+k-j}+(-1)^{k/2}C_{n+k-k/2} \\
&= \sum_{j=0}^{\lfloor k/2\rfloor}(-1)^j\binom{k-j}{j}C_{n+k-j},
\end{aligned}$$

as desired. Second, we complete the proof by induction by showing that if $k \geq 3$ is odd and the general result is true for $0, 1, 2, \ldots, k-1$, then it is true for k. For $k \geq 3$ odd and $n = 1, 2, \ldots,$

$$\begin{aligned}
C(n|k) &= C(n+1|k-1)-C(n|k-1)-C(n+1|k-3) \\
&= \sum_{j=0}^{\frac{k-1}{2}}(-1)^j\binom{k-1-j}{j}C_{n+k-j}-\sum_{j=0}^{\frac{k-1}{2}}(-1)^j\binom{k-1-j}{j}C_{n+k-1-j} \\
&\quad -\sum_{j=0}^{\frac{k-3}{2}}(-1)^j\binom{k-3-j}{j}C_{n+k-2-j}.
\end{aligned}$$

Shifting the index of summation by one in the second sum and by two in the third sum, the previous line becomes

$$\begin{aligned}
C(n|k) &= \sum_{j=0}^{\frac{k-1}{2}}(-1)^j\binom{k-1-j}{j}C_{n+k-j}+\sum_{j=1}^{\frac{k+1}{2}}(-1)^j\binom{k-j}{j-1}C_{n+k-j} \\
&\quad -\sum_{j=2}^{\frac{k+1}{2}}(-1)^j\binom{k-1-j}{j-2}C_{n+k-j}.
\end{aligned}$$

Removing the first two terms from the first sum, the first and last term from the second sum, and the last term from the last sum yields

$$\begin{aligned}
C(n|k) &= C_{n+k} - (k-2)C_{n+k-1} - C_{n+k-1} + (-1)^{\frac{k+1}{2}} C_{n+\frac{k-1}{2}} - (-1)^{\frac{k+1}{2}} C_{n+\frac{k-1}{2}} \\
&\quad + \sum_{j=2}^{\frac{k-1}{2}} (-1)^j \left[\binom{k-1-j}{j} + \binom{k-j}{j-1} - \binom{k-1-j}{j-2} \right] C_{n+k-j} \\
&= C_{n+k} - (k-1)C_{n+k-1} + \sum_{j=2}^{\frac{k-1}{2}} (-1)^j \binom{k-j}{j} C_{n+k-j} \\
&= \sum_{j=0}^{\lfloor k/2 \rfloor} (-1)^j \binom{k-j}{j} C_{n+k-j}.
\end{aligned}$$

Appendix M

Monte Carlo Simulation for Covariance Estimation Between Customers a and b with k Customers Present at Time Zero

This code estimates the covariance between the sojourn times of customers i and j in an $M/M/1$ queue with k customers present at time zero, where $i, j \leq k$. The code substantiates the results from Theorem 6.4.

```
# ---------------------------------------------------------------
# Monte Carlo simulation for the sojourn time  covariance between
# customers i, j <= k, where k customers are present at time zero.
#
# Name            : kcov.txt
# Author          : Billy Kaczynski
# Language        : R/S-Plus
# Latest Revision : 03/20/09
# ---------------------------------------------------------------

N  <- 1000000
```

Appendix M. Monte Carlo Simulation for Covariance Estimation Between Customers a and b with k Customers Present at Time Zero

```
T1 <- rexp(N, 5)
T2 <- T1 + rexp(N, 5)
T3 <- T2 + rexp(N, 5)
T4 <- T3 + rexp(N, 5)
T5 <- T4 + rexp(N, 5)
cov(T1, T2)
cov(T1, T3)
cov(T1, T4)
cov(T1, T5)
cov(T2, T3)
cov(T2, T4)
cov(T2, T5)
cov(T3, T4)
cov(T3, T5)
cov(T4, T5)
```

Appendix N

Calculating Covariance Between Customers in an $M/M/1$ Queue with k Customers Present at Time Zero

The list of procedures presented here calculates the covariance between two specific customers in an $M/M/1$ queue, where k customers are present at time zero and n additional customers arrive and process through the system after time zero, without regard to the usual traffic intensity requirement $\rho < 1$. Some procedures mentioned but not included have already been provided in Appendix E.

N.1 kcases(n, k)

```
# kcases(n, k)
# ---------------------------------------------------------------
# Generates all possible arrival/departure sequences for n
# customers in an M/M/1 queue with k customers initially present.
# Resulting list of sequences consists of 1's and -1's, where a 1
# is an arrival and a -1 is a departure. The sequences are
# returned in the matrix C. The procedure calls cases(n + k)
# which subsequently calls ini(n + k) to initialize the first
# sequence in the matrix, then uses the procedures swapa(n, A),
# swapb(n, A), and okay(n, A) to create the remaining sequences
# according to a prefix-shift algorithm.  C is then simplified by
# deleting the rows where the first k entries are not 1s.
# Furthermore, the first k columns are also deleted since they
# must all contain 1s to represent the arrivals of the k
# customers present at time zero. For each row in the resulting
# matrix, an associated path matrix can be generated via the
# procedure kpath(n, k, A).
#
# Name            : kcases.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/07/09
#-----------------------------------------------------------------

kcases := proc(n, k)
  local i, j, C;
  C := cases(n + k);
  j := 1;
  while j < RowDimension(C) + 1 do
    if (add(C[j, i], i = 1 .. k) <> k) then
      C := DeleteRow(C, j);
    else
      j := j + 1;
    fi:
  od:
  C := DeleteColumn(C, 1 .. k);
  return C;
end:
```

N.2 kpath(n, k, A)

```
# kpath(n, k, A)
#------------------------------------------------------------------
# Creates a path matrix of size (n + k + 1) by (n + 1), where 1s
# represent the arrival/service sequence for a given row of the
# case matrix C.  All other elements in the path matrix are 0.
# The path starts at the lower-left corner of the matrix and
# moves to the upper-right corner.  The first leg of the path is
# either the arrival of a customer represented by the entry in
# the [n + k + 1, 2] position or a departure represented by the
# entry in the [n + k, 1] position.  A 1 to the right of the
# previous 1 signifies an arrival, while a 1 above the previous 1
# signifies a service completion.  The procedure requires the
# arguments n, the number of customers processing through the
# system arriving after time 0, k, the number of customers
# present at time 0, and A, a row from the case matrix C.
#
# Name            : kpath.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/07/09
# ------------------------------------------------------------------

kpath := proc(n, k, A)
   local j, row, col, pat;
   row := n + k + 1;
   col := 1;
   pat := Matrix(n + k + 1, n + 1);
   pat[n + k + 1, 1] := 1;
   for j from 1 to (2 * n + k) do
     if (A[1, j] = 1) then
       col := col + 1;
       pat[row, col] := 1;
     else
       row := row - 1;
       pat[row, col] := 1;
     fi;
   od:
   return pat;
end:
```

N.3 kCprime(n, k, C)

```
# kCprime(n, k, C)
#----------------------------------------------------------------
# Produces the matrix defined as C', that is the distribution
# segment matrix where each row represents the distribution
# segments for the case represented by the corresponding row
# in the case matrix C.  The elements of C' are limited to a
# 0, 1, and 2, where 0 implies no sojourn time distribution
# segment due to an emptying of the system, 1 implies a
# competing risk of an arrival or completion of service and is
# distributed exponential(lambda + mu), and a 2 implies a
# service completion distribution leg which is distributed
# exponential(mu).  The matrix C' has the same number of rows as
# C, and 2(n + 1) + k columns.  Cprime(n, k, C) calls
# kpath(n, k, A) and uses the path matrix to determine the
# appropriate probability distribution function segments.
# The procedure requires the arguments n, number of customers
# arriving after time 0, k, the number of customers present at
# time 0, and C, the case matrix.
#
# Name            : kCprime.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/07/09
# ---------------------------------------------------------------

kCprime := proc(n, k, C)
   local prime, i, pat, dist, j, row, col;
   prime := Matrix(RowDimension(C), 2 * n + k);
   for i from 1 to RowDimension(C) do
     row := n + k + 1;
     col := 1;
     pat := kpath(n, k, C[[i], 1 .. -1]);
     dist := Matrix(1, 2 * n + k);
     for j from 1 to (2 * n + k) do
       if (pat[row - 1, col] = 1) and (col < n + 1) then
         row := row - 1;
         dist[1, j] := 1;
       elif (pat[row - 1, col] = 1) and (col = n + 1) then
         row := row - 1;
```

```
          dist[1, j] := 2;
        elif (pat[row, col + 1] = 1) and (row + col > n + 2) then
          col := col + 1;
          dist[1, j] := 1;
        else
          col := col + 1;
          dist[1, j] := 0;
        fi;
      od;
      prime[[i], 1 .. -1] := dist;
    od;
    return prime;
  end:
```

N.4 kcaseprob(n, k, P)

```
# kcaseprob(n, k, P)
#-----------------------------------------------------------------
# Computes the probability associated with a given row of the
# case matrix C as represented by the path created by
# kpath(n, k, A).  Similar to how C' identifies the appropriate
# distribution segments along the path, kcaseprob(n, k, P)
# identifies the appropriate probability for each leg of the
# path based on whether a competing risk occurs.  Requires the
# arguments n, number of customers arriving after time 0, k,
# the number of customers present at time 0, and P, the path
# of a given case.  Returns the probability of the case passed
# to the procedure.
#
# Name            : kcaseprob.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/07/09
# ----------------------------------------------------------------

kcaseprob := proc(n, k, P)
   global X, Y;
   local p, j, row, col;
   p := 1;
   row := n + k + 1;
   col := 1;
   for j from 1 to (2 * n + k) do
     if (P[row - 1, col] = 1) and (col < n + 1) then
       row := row - 1;
       p := p * 1 / Mean(Y) / (1 / Mean(X) + 1 / Mean(Y));
     elif (P[row - 1, col] = 1) and (col = n + 1) then
       row := row - 1;
     elif (P[row, col + 1] = 1) and (row + col > n + 2) then
       col := col + 1;
       p := p * 1 / Mean(X) / (1 / Mean(X) + 1 / Mean(Y));
     else col := col + 1;
     fi;
   od:
   return p;
end:
```

N.5 kprobvec(n, k)

```
# kprobvec(n, k)
#-----------------------------------------------------------------
# Uses the procedure kcaseprob(n, k, P) successively to build a
# vector of probabilities, one for each case of the C matrix.
# This vector has length C.  Requires the arguments n, the number
# of customers arriving after time 0 and k, the number of
# customers present at time 0.
#
# Name            : kprobvec.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/07/09
# ---------------------------------------------------------------

kprobvec := proc(n, k)
   local i, p, C;
   C := kcases(n, k);
   p := Vector(RowDimension(C));
   for i from 1 to RowDimension(C) do
     p[i] := kcaseprob(n, k, kpath(n, k, C[[i], 1 .. -1]));
   od:
   return p;
 end:
```

N.6 kTmat(a, b, k, A)

```
# kTmat(a, b, k, A)
#-----------------------------------------------------------------
# Creates the 2 by 2 matrix for determining whether selected
# customer sojourn times are independent and whether the customer
# index is less than or equal to k, the number of customers
# present at time 0. Also provides information on the required
# distribution segments for calculating the joint distribution
# between two customers.  Requires the arguments a, the index of
# the first customer of interest in the system, b, the index of
# the second customer of interest in the system, k, the number of
# customers present at time 0, and A, a single row of the case
# matrix C, representing a given case.  It uses this row of C to
# identify the start and finish indices for customers a and b.
# If these indices overlap, sojourn times are dependent, if they
# do not overlap the sojourn times are independent.
#
# Name            : kTmat.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/08/09
# ----------------------------------------------------------------

kTmat := proc(a, b, k, A)
  local sta, fina, stb, finb, indexa, indexb, i, T;
  indexa := 0;
  indexb := 0;
  if a <= k then
    sta := 1;
  else
    for i from 1 to ColumnDimension(A) do
      if A[1, i] = 1 then
        indexa := indexa + 1;
        if indexa = a - k then sta := i + 1 fi:
      fi:
    od:
  fi:
  indexa := 0;
  if b <= k then
    stb := 1;
  else
     for i from 1 to ColumnDimension(A) do
      if A[1, i] = 1 then
        indexa := indexa + 1;
        if indexa = b - k then stb := i + 1 fi:
      fi:
    od:
  fi:
  for i from 1 to ColumnDimension(A) do
    if A[1, i] = -1 then
      indexb := indexb - 1;
      if indexb = -a then fina := i fi:
      if indexb = -b then finb := i fi:
    fi:
  od:
  T := Matrix(2, 2, [[sta, fina], [stb, finb]]);
  return T;
end:
```

N.7 kinde(a, b, k, T, A)

```
# kinde(a, b, k, T, A)
#-------------------------------------------------------------
# Calculates the case-specific joint cumulative distribution
# function for customers a and b whose sojourn times are
# independent by multiplying the CDFs of each customer.  The
# individual customer CDFs are calculated by determining the
# type and number of distribution legs using the arguments a,
# the index of the first customer of interest, b, the index of the
# second customer of interest, T, the resulting matrix from the
# call kTmat(a, b, k, A), and A, the row of C' associated with the
# specific case.  The CDF forms for each case arise from
# appropriately defined random variables in APPL. The procedure
# returns the joint cumulative distribution function in a
# vector of length two, where both elements are identical in
# order to match the piecewise result for customers with
# dependent sojourn times.
#
# Name            : kinde.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/15/09
# -------------------------------------------------------------

kinde := proc(a, b, k, T, A)
  options remember;
  global X, Y;
  local i, dist1, dist2, jcdf, expa, expb;
  expa := 0;
  expb := 0;
  for i from 0 to (T[1, 2] - T[1, 1]) do
    if A[1, T[1, 1] + i] = 1 then
      expa := expa + 1;
    elif A[1, T[1, 1] + i] = 2 then
      expb := expb + 1;
    fi:
  od:
  if expb = 0 then dist1 := ErlangRV(1 / Mean(X) + 1 /
                            Mean(Y), expa)
  elif expa = 0 then dist1 := ErlangRV(1 / Mean(Y), expb)
  else dist1 := [[unapply(conv(expa, expb), w)], [0, infinity],
                ["Continuous", "PDF"]];
  fi:
  expa := 0;
  expb := 0;
  for i from 0 to (T[2, 2] - T[2, 1]) do
    if A[1, T[2, 1] + i] = 1 then
      expa := expa + 1;
    elif A[1, T[2, 1] + i] = 2 then
      expb := expb + 1;
    fi:
  od:
  if expb = 0 then dist2 := ErlangRV(1 / Mean(X) + 1 /
                            Mean(Y), expa)
  elif expa = 0 then dist2 := ErlangRV(1 / Mean(Y), expb)
  else dist2 := [[unapply(conv(expa, expb), w)], [0, infinity],
                ["Continuous", "PDF"]];
  fi:
  jcdf := apply(op(CDF(dist1)[1]), t[a]) *
          apply(op(CDF(dist2)[1]), t[b]);
  return Matrix([jcdf, jcdf]);
end:
```

N.8 kdep(a, b, k, T, A)

```
# kdep(a, b, k, T, A)
#---------------------------------------------------------------
# Calculates the case-specific joint cumulative distribution
# function for customers a and b whose sojourn times are
# dependent by conditioning on the overlap distribution
# segment(s).  The customer sojourn time segments are divided
# into their associated independent and dependent (overlap)
# portions.  This amounts to three segments, customer a's
# independent portion defined as dist1, customer b's
# independent portion defined as dist2, and the dependent
# overlap portion defined as dist3.  The joint cumulative
# distribution function has two pieces, for the cases when
# t[a] < t[b] and t[b] < t[a]. When a and b are both less than
# or equal to k, only two distribution segments arise, dist1
# and dist2, because both sojourn times start at time 0.
# Therefore the sojourn time T[b] > T[a], and the resulting
# joint cumulative distribution function has only a single piece.
#
# Name            : kdep.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/15/09
# --------------------------------------------------------------

kdep := proc(a, b, k, T, A)
  options remember;
  global X, Y;
  local i, expa, expb, dist1, dist2, dist3, jcdftop, jcdfbot;
  expa := 0;
  expb := 0;
  if ((a <= k) and (b <= k)) then
    for i from 0 to (T[1, 2] - T[1, 1]) do
      if A[1, T[1, 1] + i] = 1 then
        expa := expa + 1;
      elif A[1, T[1, 1] + i] = 2 then
        expb := expb + 1;
      fi:
    od:
    if expb = 0 then dist1 := ErlangRV(1 / Mean(X) + 1 / Mean(Y),
                                   expa)
    elif expa = 0 then dist1 := ErlangRV(1 / Mean(Y), expb)
    else dist1 := [[unapply(conv(expa, expb), w)], [0, infinity],
                  ["Continuous", "PDF"]];
    fi:
    expa := 0;
    expb := 0;
    for i from 1 to (T[2, 2] - T[1, 2]) do
      if A[1, T[1, 2] + i] = 1 then
        expa := expa + 1;
      elif A[1, T[1, 2] + i] = 2 then
        expb := expb + 1;
      fi:
    od:
    if expb = 0 then dist2 := ErlangRV(1 / Mean(X) + 1 / Mean(Y),
                                   expa)  ;
    elif expa = 0 then dist2 := ErlangRV(1 / Mean(Y), expb);
    else dist2 := [[unapply(conv(expa, expb), w)], [0, infinity],
                  ["Continuous", "PDF"]];
    fi:
    jcdftop := simplify(int(int(apply(op(PDF(dist2)[1]), y) *
                apply(op(PDF(dist1)[1]), x), y = 0 ..
                (t[b] - x)), x = 0 .. t[a]));
    return Matrix([jcdftop]);
  else
    for i from 1 to (T[2, 1] - T[1, 1]) do
        if A[1, T[1, 1] + i - 1] = 1 then
          expa := expa + 1;
        elif A[1, T[1, 1] + i - 1] = 2 then
          expb := expb + 1;
        fi:
      od:
    fi:
    if expb = 0 then dist1 := ErlangRV(1 / Mean(X) + 1 / Mean(Y),
                                   expa);
    elif expa = 0 then dist1 := ErlangRV(1 / Mean(Y), expb);
    else dist1 := [[unapply(conv(expa, expb), w)], [0, infinity],
                  ["Continuous", "PDF"]];
    fi:
    expa := 0;
    expb := 0;
    for i from 1 to (T[2, 2] - T[1, 2]) do
```

```
      if A[1, T[1, 2] + i] = 1 then
        expa := expa + 1;
      elif A[1, T[1, 2] + i] = 2 then
        expb := expb + 1;
      fi:
    od:
    if expb = 0 then dist2 := ErlangRV(1 / Mean(X) + 1 / Mean(Y),
                                       expa);
    elif expa = 0 then dist2 := ErlangRV(1 / Mean(Y), expb);
    else dist2 := [[unapply(conv(expa, expb), w)], [0, infinity],
                  ["Continuous", "PDF"]];
    fi:
    expa := 0;
    expb := 0;
    for i from 0 to (T[1, 2] - T[2, 1]) do
      if A[1, T[2, 1] + i] = 1 then
        expa := expa + 1;
      elif A[1, T[2, 1] + i] = 2 then
         expb := expb + 1;
      fi:
    od:
    if expb = 0 then dist3 := ErlangRV(1 / Mean(X) + 1 / Mean(Y),
                                       expa);
    elif expa = 0 then dist3 := ErlangRV(1 / Mean(Y), expb);
    else dist3 := [[unapply(conv(expa, expb), w)], [0, infinity],
                  ["Continuous", "PDF"]];
    fi:
    jcdftop := int(apply(op(CDF(dist1)[1]), t[a] - y) *
               apply(op(CDF(dist2)[1]), t[b] - y) *
               apply(op(PDF(dist3)[1]), y), y = 0 .. t[a]);
    jcdfbot := int(apply(op(CDF(dist1)[1]), t[a] - y) *
               apply(op(CDF(dist2)[1]), t[b] - y) *
               apply(op(PDF(dist3)[1]), y), y = 0 .. t[b]);
    return Matrix([jcdftop, jcdfbot]);
end:
```

N.9 kjcdf(a, b, k, n)

```
# kjcdf(a, b, k, n)
#-------------------------------------------------------------------
# Creates the case joint cumulative distribution functions in a
# matrix by calling the procedure kTmat(a, b, k, A) and depending
# on the structure returned, calls the procedures
# kinde(a, b, k, T, A) or kdep(a, b, k, T, A) to generate the
# appropriate case-wise joint cumulative distribution function.
# Requires arguments a, the index of the first customer of interest,
# b, the index of the second customer of interest, and n, the number
# of customers arriving after time 0, and k, the number of customers
# present at time 0.
#
# Name            : kjcdf.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/15/09
# ------------------------------------------------------------------

kjcdf := proc(a, b, k, n)
  local C, i, dist, T;
  C := kcases(n, k);
  dist := Matrix(RowDimension(C), 2);
  for i from 1 to RowDimension(C) do
    T := kTmat(a, b, k, C[[i], 1 .. -1]);
    if T[1, 2] < T[2, 1] then
      dist[[i], 1 .. -1] := kinde(a, b, k, T, kCprime(n, k, C[[i],
                            1 .. -1]));
    else
      dist[[i], 1 .. -1] := kdep(a, b, k, T, kCprime(n, k, C[[i],
                            1 .. -1]));
    fi:
  od:
  return dist;
end:
```

N.10 kCov(X, Y, a, b, n, k)

```
# kCov(X, Y, a, b, n, k)
#-----------------------------------------------------------------
# Mixes the results returned by kprobvec(n, k) and
# kjpdf(a, b, n, k) to compute the joint cumulative distribution
# function encompassing all cases.  Differentiates the results to
# produce the piecewise joint probability distribution function.
# Calls Queue(n, k, s) to find the appropriate expected values
# for the customers of interest, then uses the expected values,
# along with the expected value E(T[a]T[b]) found using the joint
# probability distribution function, to compute the covariance as
# Cov(T[a], T[b]) = E(T[a]T[b]) - E(T[a])E(T[b]).  Requires the
# arguments X, the distribution of time between arrivals in the
# APPL list-of-lists format, Y, the service time distribution in
# the list-of-lists format, a, the index of the first customer of
# interest, and b, the index of the second customer of interest
# (a < b), n, the number of customers arriving after time 0, and k,
# the number of customers present at time 0.
#
# Name            : kcov.mw
# Author          : Billy Kaczynski
# Language        : MAPLE 9
# Latest Revision : 03/17/09
# -----------------------------------------------------------------
kCov := proc(X, Y, a, b, n, k)
  local JPDFMAT, PVEC, JPDF, fta, ftb, Etatb, Eta, Etb, Cov, fa,
        fb, ftab;
    JPDFMAT := kjcdf(a, b, k, n);
    PVEC := kprobvec(n, k);
    JPDF := Transpose(JPDFMAT) . PVEC;
    if ((a <= k) and (b <= k)) then
      ftab := simplify(diff(diff(JPDF[1], t[a]), t[b]));
      Etatb := int(int(t[a]*t[b]*ftab, t[a] = 0 .. t[b]),
               t[b] = 0 .. infinity);
      Eta := int(t[a] * int(ftab, t[b] = t[a] .. infinity),
             t[a] = 0 .. infinity);
      Etb := int(t[b] * int(ftab, t[a] = 0 .. t[b]),
             t[b] = 0 .. infinity);
      Cov := Etatb - Eta * Etb;
      return Cov;
    else
      fta := simplify(diff(diff(JPDF[1], t[a]), t[b]));
      ftb := simplify(diff(diff(JPDF[2], t[a]), t[b]));
      Etatb := int(int(t[a] * t[b] * fta, t[a] = 0 .. t[b]),
             t[b] = 0 .. infinity) + int(int(t[a] * t[b] *
             ftb, t[b] = 0 .. t[a]), t[a] = 0 .. infinity);
      fa := simplify(int(fta, t[b] = t[a]..infinity) + int(ftb,
            t[b] = 0..t[a]));
      Eta := int(t[a] * fa, t[a] = 0 .. infinity);
      fb := simplify(int(fta, t[a] = 0 .. t[b]) + int(ftb,
            t[a] = t[b] .. infinity));
      Etb := int(t[b] * fb, t[b] = 0 .. infinity);
      Cov := Etatb - Eta * Etb;
      return Cov;
    fi:
end:
```

Bibliography

[1] J. Abate and W. Whitt. Transient behavior of the $M/M/1$ queue via Laplace transforms. *Advances in Applied Probability*, 20(1):145–178, 1988.

[2] J. Banks, J.S. Carson, B.L. Nelson, and D.M. Nicol. *Discrete-event system simulation.* Prentice–Hall, 2001.

[3] F. Benford. The law of anomalous numbers. *Proceedings of the American Philosophical Society Held at Philadelphia for Promoting Useful Knowledge*, 78(4):551, 1938.

[4] A.W. Bowman and A. Azzalini. *Applied smoothing techniques for data analysis: The kernel approach with S-Plus illustrations.* Oxford University Press, 1997.

[5] P. Bratley, B.L. Fox, and L.E. Schrage. *A guide to simulation.* Springer–Verlag, 1987.

[6] I.W. Burr. The effect of non-normality on constants for $\overline{X}$ and R charts. *Industrial Quality Control*, 23(11):563–569, 1967.

[7] I.W. Burr. On a general system of distributions. III. The sample range. *Journal of the American Statistical Association*, 63:636–643, 1968.

[8] I.W. Burr. *Statistical quality control methods.* CRC Press, 1976.

[9] L. Devroye. *Non-uniform random variate generation.* Springer–Verlag, 1986.

[10] L. Devroye and L. Györfi. Nonparametric density estimation. The L–1 view. 1985.

[11] J.H. Drew, D.L. Evans, A.G. Glen, and L.M. Leemis. *Computational probability: Algorithms and applications in the mathematical sciences.* Springer, 2007.

[12] B. Efron and R.J. Tibshirani. An introduction to the bootstrap. *Monographs on Statistics and Applied Probability*, 57:1–177, 1993.

[13] G.S. Fishman, R. Durrett, R.B. Myerson, M.A. Bean, E.P.C. Kao, J.J. Higgins, S. Keller-McNulty, and R.L. Scheaffer. *A first course in Monte Carlo.* Thomson Brooks/Cole, 2006.

[14] A.V. Gafarian, C.J. Ancker Jr., and T. Morisaku. The problem of the initial transient in digital computer simulation. *Proceedings of the 76 Bicentennial Conference on Winter Simulation*, 49–51, 1976.

[15] C. Genest and B. Rémillard. Discussion of copulas: Tales and facts, by Thomas Mikosch. *Extremes*, 9(1):27–36, 2006.

[16] A.G. Glen, D.L. Evans, and L.M. Leemis. APPL: A probability programming language. *The American Statistician*, 55(2):156–166, 2001.

[17] C. Hagwood. An application of the residue calculus: The distribution of the sum of nonhomogeneous gamma variates. *The American Statistician*, 63(1):37–39, 2009.

[18] H.O. Hartley and E.S. Pearson. Moment constants for the distribution of range in normal samples. *Biometrika*, 38(3–4):463–464, 1951.

[19] T.P. Hill. A statistical derivation of the significant-digit law. *Statistical Science*, 10:354–363, 1995.

[20] T.P. Hill. Base-invariance implies Benford's law. *Proceedings of the American Mathematical Society*, 123(3):887–895, 1995.

[21] F.S. Hillier and G.J. Lieberman. *Introduction to operations research.* McGraw–Hill, 2005.

[22] R.V. Hogg, A.T. Craig, and J. McKean. *Introduction to mathematical statistics.* Macmillan, 2005.

[23] W. Hörmann and J. Leydold. Random-number and random-variate generation: automatic random variate generation for simulation input. In *Proceedings of the 32nd conference on Winter simulation.* Society for Computer Simulation International San Diego, CA, USA, 675–682, 2000.

[24] M.E. Johnson. *Multivariate statistical simulation.* Wiley, 1987.

[25] W.D. Kelton. Transient exponential–Erlang queues and steady-state simulation. *Communications of the ACM*, 28(7):741–749, 1985.

[26] W.D. Kelton and A.M. Law. The transient behavior of the $M/M/s$ queue, with implications for steady-state simulation. *Operations Research*, 33(2):378–396, 1985.

[27] L. Kleinrock. *Queueing systems.* John Wiley & Sons, 1975.

[28] R.J. Larsen and M.L. Marx. *An introduction to mathematical statistics and its applications.* Prentice–Hall, 2006.

[29] A. Law. *Simulation modelling and analysis.* McGraw–Hill, 2007.

[30] A.M. Law. A comparison of two techniques for determining the accuracy of simulation output. Technical Report 75–11, University of Wisconsin at Madison, 1975.

[31] L.M. Leemis and S.K. Park. *Discrete-event simulation: A first course.* Prentice–Hall, 2006.

[32] L.M. Leemis, B.W. Schmeiser, and D.L. Evans. Survival distributions satisfying Benford's law. *The American Statistician*, 54(4):236–241, 2000.

[33] L.M. Leemis and K.S. Trivedi. A comparison of approximate interval estimators for the binomial parameter. *The American Statistician*, 50(1):63–68, 1996.

[34] P. Leguesdron, J. Pellaumail, G. Rubino, and B. Sericola. Transient analysis of the $M/M/1$ queue. *Advances in Applied Probability*, 25(3):702–713, 1993.

[35] E. Ley. On the peculiar distribution of the U.S. stock indexes' digits. *The American Statistician*, 50(4):311–313, 1996.

[36] J. Lieblein and M. Zelen. Statistical investigation of the fatigue life of deep-groove ball bearings. *Journal of Research of the National Bureau of Standards*, 57(5):273–316, 1956.

[37] A.T. McKay and E.S. Pearson. A note on the distribution of range in samples of n. *Biometrika*, 25(3-4):415–420, 1933.

[38] W.R. Mebane Jr. Election forensics: Vote counts and benford's law. *Summer Meeting of the Political Methodology Society, University of California–Davis, July*, 2006.

[39] T. Morisaku. *Techniques for data-truncation in digital computer simulation.* PhD thesis, University of Southern California, Los Angeles, 1976.

[40] S. Newcomb. Note on the frequency of use of the different digits in natural numbers. *American Journal of Mathematics*, 4(1/4):39–40, 1881.

[41] M. Nigrini. A taxpayer compliance application of Benford's law. *Journal of the American Taxation Association*, 18(1):72–91, 1996.

[42] M. Nigrini and W. Wood. Assessing the integrity of tabulated demographic data, preprint, University of Cincinnati and St Marys University. 1995.

[43] A.R. Odoni and E. Roth. Empirical investigation of the transient behavior of stationary queueing systems. *Operations Research*, 31(3):432–455, 1983.

[44] A.B. Owen. *Empirical likelihood.* Chapman & Hall/CRC, 2001.

[45] P. R. Parthasarathy. A transient solution to an $M/M/1$ queue: A simple approach. *Advances in Applied Probability*, 19(4):997–998, 1987.

[46] C.D. Pegden and M. Rosenshine. Some new results for the $M/M/1$ queue. *Management Science*, 28(7):821–828, 1982.

[47] R.S. Pinkham. On the distribution of first significant digits. *The Annals of Mathematical Statistics*, 32(4):1223–1230, 1961.

[48] R.J. Rodriguez. First significant digit patterns from mixtures of uniform distributions. *The American Statistician*, 58(1):64–71, 2004.

[49] F. Ruskey and A. Williams. Generating balanced parentheses and binary trees by prefix shifts. *Proceedings of the 12th Computing: The Australasian Theory Symposium (CATS2008), CRPIT*, 77:107–115, 2008.

[50] W.A. Shewhart. *Economic control of quality of manufactured product.* American Society for Quality Control, 1980.

[51] B.W. Silverman. *Density estimation for statistics and data analysis.* Chapman & Hall/CRC, 1986.

[52] P. Tadikamalla, M. Banciu, and D. Popescu. An improved range chart for normal and long-tailed symmetrical distributions. *Naval Research Logistics*, 55(1):91, 2008.

[53] M.S. Taylor and J.R. Thompson. A data based algorithm for the generation of random vectors. *Computational Statistics and Data Analysis*, 4(2):93–101, 1986.

[54] L.H.C. Tippett. On the extreme individuals and the range of samples. *Biometrika*, 17:364, 1925.

[55] M.A.F. Wagner and J.R. Wilson. Graphical interactive simulation input modeling with bivariate Bézier distributions. *ACM Transactions on Modeling and Computer Simulation*, 5(3):163–189, 1995.

[56] S. Weisberg. *Applied linear regression.* Wiley, 1980.

[57] D.J. Wheeler. *Normality and the process behavior chart.* SPC Press, 2000.

[58] W.L. Winston. *Operations research: Applications and algorithms.* Thompson, 2004.

Vita

William H. Kaczynski is a Lieutenant Colonel in the United States Army, and is currently assigned as an assistant professor in the Department of Mathematical Sciences at the United States Military Academy located in West Point, New York. He was commissioned in the U.S. Army aviation branch as a Second Lieutenant in 1992, earning a Bachelor of Science degree in Economics from the United States Military Academy. He also holds a Master of Science degree in Operations Research from the Georgia Institute of Technology from 2002. He is a career Army officer with 17 years of service in varied Army leadership positions. He has been assigned at Fort Rucker, AL, Fort Bliss, TX, Fort Carson, CO, Camp Page, Republic of South Korea, the Georgia Institute of Technology, and the College of William & Mary. He is married to the former Natalie E. Moreland, and they have two daughters, Madison and Haley.