International Series in
Operations Research & Management Science

Barry L. Nelson
Linda Pei

Foundations and Methods of Stochastic Simulation

A First Course

Second Edition

MOREMEDIA

Springer

Barry L. Nelson • Linda Pei

Foundations and Methods of Stochastic Simulation

A First Course

Second Edition

Springer

Barry L. Nelson
Department of Industrial Engineering and Management Sciences
Northwestern University
Evanston, IL, USA

Linda Pei
Department of Industrial Engineering and Management Sciences
Northwestern University
Evanston, IL, USA

ISSN 0884-8289 ISSN 2214-7934 (electronic)
International Series in Operations Research & Management Science
ISBN 978-3-030-86193-3 ISBN 978-3-030-86194-0 (eBook)
https://doi.org/10.1007/978-3-030-86194-0

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To my undergraduate, master's, Ph.D. and postdoctoral students, for pushing me and making it all worthwhile—Barry

To my fellow Ph.D. nerds SR, RV, RR, AS, MS, EW, DD, PBC, GW, CP, EE, JL, and LK: thanks for your warmth, wit, and humanity—Linda

Preface to the Second Edition

Despite the addition of a significant amount of new material, our approach remains "to be *concise, precise, and integrated*, leaving a lot of room for the instructor to expand on areas of interest or importance to them" while still being "*complete*." We (and it now really is "we") strive to present foundational ideas that are unlikely to go out of date, while also providing tools for solving real problems.

We believe, now more than ever, that graduate students and many undergraduates need to know how to program simulations in a general-purpose language, both to facilitate complete control of the model and (sometimes) to be computationally efficient. The first edition of the book employed VBA because it was and is widely available, and it is an easy first programming language. The second edition shifts to Python, which is free, relatively easy to learn, and extensively used in all sorts of applications. Rather than directly port VBASim to Python, we took care to rethink our approach and to exploit useful features of Python (in particular its object-oriented powers) without depending on many special-purpose libraries. Readers of the first edition will recognize many aspects of VBASim in PythonSim, but will also see distinct differences. The philosophy of employing a compact collection of classes and functions that capture the key features of discrete-event simulation, which can be fully understood and easily modified or extended, and which support programming realistic simulations, still holds.

Related to computation, a big change from when the first edition was published in 2013, is the availability of cheap, high-performance, and increasingly easy-to-use parallel computing. The first author has been known to say, "Anyone doing simulation research or practice, who is not thinking about how it will parallelize, is missing the point." As a result, Chap. 5 was expanded from *two* views of simulation to *three* with the third being simulation as computation. Then scattered throughout the book are discussions about how parallelization affects various aspects simulation.

Chapter 5 also includes new material on large-deviations properties and Gaussian processes, both critical to modern simulation research. Sensitivity analysis plays an expanded role in the second edition, affecting both Chaps. 7 and 9. Chapter 9 results from splitting Chap. 8 from the first edition to accommodate new material on simulation optimization (including Bayesian approaches and exploiting parallel simulation), sensitivity analysis, and change of measure (which includes "importance sampling"). Chapter 10 (formerly Chap. 9) adds suggestions for creating reproducible simulation results. And, of course, new exercises have been added to many chapters. The book's website is still at http://users.iems.northwestern.edu/~nelsonb/IEMS435/.

We thank David Eckman, Shane Henderson, Susan Hunter, and Chang-han Rhee for advice, and the many folks who provided (always polite) feedback on the first edition, as well as the stochastic simulation community in general that has supported our efforts for many years.

Evanston, IL, USA Barry L. Nelson

Evanston, IL, USA Linda Pei

Preface to the First Edition

As is often the case, this book grew out of a course. The interesting part is that the course grew out of this story: I was helping one of our graduate students, Viji Krishnamurthy, whose research involved developing rules for the use of flexible workers in a repair and maintenance environment (Iravani & Krishnamurthy, 2007). Using Markov chain analysis, Viji had derived some optimal strategies for simple systems and now wanted to test their robustness for more realistic problems. This is where simulation came in. Viji had taken the typical first simulation modeling course that used a commercial simulation product and an advanced course that focused only on design and analysis, but not model building. She (and I) spent hours trying to trick a commercial product into simulating the complex worker allocation rules that she wanted to test. Commercial simulation environments make modeling easy by including the sort of system features that users typically want. Unfortunately, making it easy to model typical features can make it difficult to represent something different, and research is always about something different.

Finally, in frustration, I handwrote three pages of pseudocode to simulate exactly what Viji wanted, handed her the notes, and told her to code it up in C (which fortunately she could do). She came back the next day excited: "Now I can test anything I want, and it runs in seconds!" From this experience, IEMS 435 Introduction to Stochastic Simulation—a required course for all of our Ph.D. students—was born. IEMS 435 is not just about modeling and programming, however. Viji also needed to run well-designed experiments on her model, experiments that could provide compelling evidence for or against her analytically derived rules. So, experiment design and analysis are also a part of the IEMS 435 course, which this book was written to support.

The objectives of the book are as follows:

- To prepare students who have never had a discrete-event, stochastic simulation course to build simulations in a lower-level programming language. I am convinced that if they can do this, they can easily pick up higher-level simulation modeling environments when they need them (maybe for teaching a course as faculty).

- To prepare students to *use* simulation in their non-simulation research. This is why I emphasize actually programming simulations—which provides the greatest flexibility, control, and understanding—and experiment design and analysis.
- To prepare students to go into an advanced course on simulation methodology, including independent studies directed by their advisers. The usual first course in simulation emphasizes modeling and commercial software and is poor preparation for a research-oriented advanced course which may treat the simulation almost entirely as a mathematical object, breaking the critical connection between modeling and analysis.
- To provide a solid mathematical/statistical grounding in simulation and some (but not all) tools to solve actual problems.

The philosophy of the book is similar to Law (2007) in covering both simulation modeling and analysis, but is different in that there is no attempt to be comprehensive or survey the field. The goal is to be *concise, precise, and integrated*, leaving a lot of room for the instructor to expand on areas of interest or importance to them. The hope is that an instructor will want students to read *all* of the book to get a complete, coherent picture before jumping off into other reference texts or journal papers. To that end, I provide pointers to relevant literature. However, while not comprehensive, the book is *complete*; so, it is appropriate for students or researchers who need to learn the basics of simulation on their own without the benefit of a course. The book by Asmussen and Glynn (2007) shares some of the same objectives as this book; it is an excellent introduction to advanced simulation analysis and covers more of the topic than I do, but it does not address modeling or programming to the same extent.

The material on simulation modeling and programming, which is isolated to two chapters, uses Visual Basic for Applications (VBA) in Excel. This choice was driven by the reality that fewer and fewer graduate students come to me with programming experience. VBA/Excel is readily available, is easy to pick up quickly, and prepares students to learn Java, C++, or any other programming language later. This part of the book can be skipped for students who already know how to program simulations without compromising the remainder. Both Java and Matlab versions of Chap. 4, all of the software described in the book, and any data sets needed for exercises are available for download at the book website:

http://users.iems.northwestern.edu/~nelsonb/IEMS435/

The book should serve advanced undergraduates and graduate students. A prerequisite is a solid course in probability and statistics; statistics alone is not adequate. Although the book uses tractable stochastic process models (e.g., Markovian queues) as examples, the reader is not expected to have any background in these topics (in fact, students may find the stochastic processes course more intuitive and meaningful after having worked through this book). If students have had no experience with programming computer algorithms—for instance, in Matlab or some other programming language—then the instructor will have to supplement the book with more programming practice. Being an accomplished, or even good, programmer is not required, however.

Chapter 1 provides a concise summary of the book—except for programming—through a simple reliability problem. Chapter 2 then fills that gap by giving a quick start on simulation programming, using VBA. Chapter 3 introduces a number of tractable examples that illustrate the key issues that arise in analyzing stochastic systems via simulation; these examples recur throughout the book and so it is a must-read chapter. VBASim, a collection of VBA subs and class modules developed to support the book, is covered in Chap. 4 and is skippable if some other programming language or package will be used. Chapter 5 strengthens the connection between simulation and mathematical/numerical analysis of stochastic processes; it has the dual mission of setting up the design and analysis chapters that follow and preparing the student for more advanced courses on simulation methodology. Chapters 6–8 cover input modeling, output analysis, and experiment design, respectively, and are largely independent of the programming approach actually used to construct the simulation. The book concludes with Chap. 9, a guide to using simulation in research as opposed to using simulation to solve systems analysis problems.

What is missing? The book does not touch on the computing environment, and there are things you might want to do differently if you have, say, 500 CPUs available in a cloud computing cluster. I anticipate that by the time there is a need for a second edition, it will be easier to leverage such an environment for discrete-event, stochastic simulation, and I will add some general guidelines and recommendations. And while there is a lot of material on simulation optimization, the text is light on specific algorithms, reflecting the fact that there is no current agreement on baseline methods for all types of problems. That too will change. Finally, beyond discussing what it means, I did not do justice to the topic of validation of simulation models. A number of students and colleagues contributed to the development of the programming approach used in this book. IEMS 435 initially used Law & Kelton (2000) as the text, and Dingxi Qiu spent a summer converting all of the C code in that book into VBA. Christine Nguyen helped in the development of VBASim, the simulation support library described in this book. Feng Yang worked on a research project with me where we used VBA for simulation analysis. Lu Yu assisted with the development of the solutions manual. The Java and Matlab versions of VBASim were translated by Luis de la Torre and Weitao Duan, respectively.

I have gotten a lot of feedback. Students in IEMS 435 suffered through incomplete versions of the text, spotting errors and typos with glee. Larry Leemis provided a thorough mark-up of an early draft, and Jason Merrick taught from it. It is good to have friends, and a number of mine read and marked up sections of the nearly complete book, including Christos Alexopoulos, Bahar Biller, John Carson, Xi Chen, Ira Gerhardt, Jeff Hong, Sheldon Jacobson, Seong-Hee Kim, Jack Kleijnen, Jeremy Staum, Laurel Travis, Feng Yang, Wei Xie, Jie Xu, and Enlu Zhou. Michael Fu guided my thinking about how to develop the section on gradient estimation, and Bruce Schmeiser did the same for error estimation. Seyed Iravani, Chuck Reilly, and Ward Whitt made sure I did not mangle the message of their papers in Chap. 10. In addition to those listed above, other people whose work influenced my thinking in this book include Sigrún Andradóttir, Russell Cheng, Dave Goldsman, Shane

Henderson, David Kelton, Pierre L'Ecuyer, Lee Schruben, and Jim Wilson. Thank you all.

Last, but not least, work on this book was partially supported by National Science Foundation grant number CMMI-1068473. NSF does more with less than any federal government agency I know; let's keep it around.

Evanston, IL, USA Barry L. Nelson

Contents

Chapter 1
Why Do We Simulate?

Stochastic simulation is a method for analyzing the performance of systems whose behavior depends on the interaction of random processes, processes that can be fully characterized by probability models. Stochastic simulation is a companion to mathematical and numerical analysis of stochastic models (e.g., Nelson, 1995) and is often employed when the desired performance measures are mathematically intractable or there is no numerical approximation whose error can be bounded. Computers make stochastic simulation practical, but the method can be described independently of any computer implementation, which is what we do here.

1.1 An Example

We start with a simple, stylized example that nevertheless illustrates the key issues that we address in this book. We will use such examples throughout the text, and they have been carefully selected to exhibit the complexity of realistic problems, without being complicated to analyze or simulate. In both teaching and research, examples that illustrate complex behavior, but are not complicated to explain or analyze, are very important: they build intuition and provide a way to think through new ideas without obscuring things with a myriad of details. Realistic models may indeed be complicated, having many inputs and outputs and requiring thousands of lines of computer code to implement, so we also address modeling and programming simulations in Chaps. 2–4.

Example 1.1 (System Time to Failure TTF*).* The TTF system we consider here consists of two components that work as an active and a cold spare. The spare component becomes the active component when the (currently) active component fails, while the failed component immediately commences repair. The failed component becomes the spare when its repair is completed. Only one component at a time can

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_1

be repaired, so the system fails if both components have failed, and it is operational as long as at least one of the components is functioning. The lifetime (time to failure) of a component is equally likely to be 1, 2, 3, 4, 5, or 6 days, while repair takes exactly 2.5 days. Further, a repaired component is as good as new.

If we were interested in such a system, then we would probably want to know about its failure characteristics; for instance, its mean time to first system failure or its long-run system availability. Notice that while the failure characteristics of the components are completely specified (their lifetimes are equally likely to be from 1 to 6 days), the failure characteristics of the system as a whole—which depend on the interaction of failure and repair—are not immediately apparent. In fact, as simple as this system is it is difficult to derive these performance characteristics mathematically, while it is easy to simulate them (as we show below). This is why we simulate.

We will call those features of a system whose behavior is fully described by a probability model, such as the time to failure of a component, *inputs*. Derived quantities like the time of first system failure or system availability over some time horizon we call *outputs*.

> The stochastic simulation method consists of generating realizations of the inputs, executing the system's logic to produce outputs and estimating system performance characteristics from the outputs.

Chapter 6 addresses representing and generating inputs; Chap. 7 covers analyzing the outputs, and Chap. 8 is about designing the experiment.

Let the *state* of the TTF system at any point in time be the number of functional components, $2,1$, or 0. The *events* that can cause the state of the system to change are the failure of a component and the completion of a repair. The current state, a list of pending future events—called an *event calendar*—and a *clock* are adequate to simulate *sample paths* of system behavior, provided we have a source of randomness to generate realizations of the inputs (time to failure of a component in this example). Here we will use a physical mechanism, a fair six-sided die, as our source of randomness.

Table 1.1 shows the sample path that would occur if the first four rolls of the die were $5,3,6,1$ and we stopped at the time of the first system failure (when the state reaches 0, meaning no functional components); each roll is enclosed in a box, so $\boxed{6}$ represents a roll of the die yielding 6. The path was generated by executing the generic simulation algorithm shown in Fig. 1.1, noting that when a failure occurs the state decreases by one, while a repair increases it by one.

The approach illustrated in Table 1.1 often seems unnatural at first. For instance, why do we only keep track of the Next Failure when it is easy to figure out when each of the components will fail? And why not put the Next Repair on the calendar at time 0 since we can easily see it will be at time 7.5? The reason is the

> Lazy simulator's rule: Schedule a future event only if not doing so could cause events to occur out of chronological order.

For instance, although we could figure out when both components will fail, we are not forced to schedule the failure of the spare until that moment when the active

Table 1.1 Simulation of the TTF system until the first system failure

	System	Event calendar		
Clock	state	Next failure	Next repair	Comment
0	2	$0+\boxed{5}=5$	∞	Initialize fully operational
5	1	$5+\boxed{3}=8$	$5+2.5=7.5$	One active, one under repair
7.5	2	8	∞	Next Failure remains on calendar
8	1	$8+\boxed{6}=14$	$8+2.5=10.5$	One active, one under repair
10.5	2	14	∞	Fully functional again
14	1	$14+\boxed{1}=15$	$14+2.5=16.5$	Active fails quickly
15	0	∞	16.5	System failure

Initialize: Set the initial values of the clock and system state, and put at least one future event on the event calendar.
Advance time: Update the clock to the time of the next pending event (and remove that event from the calendar).
Update: Update the state as appropriate for the current event and schedule any new future events to occur at the current clock time plus a time increment.
Terminate: If a termination condition has occurred, then stop; otherwise go to Advance time.

Fig. 1.1 Generic simulation algorithm

component has failed. Similarly, we do not have to schedule a repair to occur until a component actually fails, even if we could work it out. On the other hand, at time 8 we *must* schedule both the Next Failure and Next Repair events because the evolution of the system from that point on depends on which of them occurs first. This rule turns out to be essential when programming complicated simulations that can have many types of events on the calendar simultaneously; in particular it avoids the need to cancel future events that become irrelevant because of another event that occurs first.

Here are some characteristics of the TTF example that are common to (nearly all) the stochastic simulations in this book:

- Simulated time (the simulation clock) jumps from event time to event time, rather than updating continuously; for this reason we call such simulations *discrete-event simulations*.
- The clock, the current state of the system, the list of future events, and the event logic are all we need to advance the simulation to the next state change.
- The simulation ends when a particular system state is reached, at a fixed simulation time, or when a particular event occurs.

1.2 Formalizing the Analysis

Table 1.1 shows one sample path of the TTF system. Let the time of first system failure be denoted by Y, and the number of functional components at time t by $S(t)$. One sample path provides one observation of Y, but $S(t)$ is a function whose value evolves over the course of the simulation. If we are interested in the average number of functional components over some simulated time horizon T, denoted by $\bar{S}(T)$, then it is a *time average* because $S(t)$ has a value at all points in time. Thus,

$$\bar{S}(T) = \frac{1}{T}\int_0^T S(t)\,\mathrm{d}t = \frac{1}{e_N - e_0}\sum_{i=1}^{N} S(e_{i-1}) \times (e_i - e_{i-1}),$$

where $0 = e_0 \le e_1 \le \cdots \le e_N = T$ are the event times in the sample path. A characteristic of continuous-time outputs, like $S(t)$, in a discrete-event simulation is that they are piecewise-constant, since they can only change values at event times.

For the simulation in Table 1.1, the observed value of Y is 15, and the observed value of $\bar{S}(T)$ at $T = 15$ is

$$\frac{1}{15-0}\big[2(5-0)+1(7.5-5)+2(8-7.5)+1(10.5-8)+2(14-10.5)$$
$$+1(15-14)\big] = \frac{24}{15}.$$

Of course, this is only one possible pair of outputs $(Y, S(Y))$. *Replications* are statistically independent repetitions of the same model, and we will often make multiple replications to improve our estimates of system performance. An important distinction is between *within-replication* and *across-replication* output data. Both Y and $S(t)$ are *within-replication* outputs. The times of system failure $Y_1, Y_2, \ldots, Y_n$, and the average number of functional components, $\bar{S}_1(T), \bar{S}_2(T), \ldots, \bar{S}_n(T)$, from n different replications are *across-replication* outputs. Results across replications are naturally i.i.d. They are independent because we roll the die anew on each replication, and identically distributed because we apply the same initial conditions and model logic to those rolls of the die.

We run simulations to estimate system performance, often to compare alternative designs for a system. We can justify using simulation-based estimators when they satisfy some version of the *strong law of large numbers (SLLN)*, a topic we address more formally in Chap. 5. Two versions of the SLLN are relevant to stochastic simulation:

1. As the number of replications n increases, it may be that

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} Y_i = \mu$$

(with probability 1), where μ can be interpreted as the the mean time to first system failure in the TTF example.

2. As the length of the replication T goes to infinity (which would make sense if we did not stop the simulation at the time of first system failure), it may be that

$$\lim_{T\to\infty} \frac{1}{T}\int_0^T S(t)\,\mathrm{d}t = \theta$$

(with probability 1), where θ can be interpreted as the long-run average number of functional components in the TTF example.

The message is that as simulation effort increases (number of replications, length of replication, or perhaps both), the simulation estimators converge in a very strong sense to some useful system performance measures. This is comforting, but in practice we stop well short of infinity and thus do not fully converge. Therefore, an important topic is measuring the remaining error when we do stop; a strength of stochastic simulation is that we can do just that by using statistical inference.

1.3 Issues and Extensions

The TTF example illustrates the basics of stochastic simulation, but not all of the problems that can arise. Consider the following:

- Suppose that the repair times are not precisely 2.5 days, but instead are 1 day with probability $1/2$, and 3 days with probability $1/2$. A fair coin is all we need to generate repair times, but what if a failure and repair are scheduled to occur at the same time (which now can clearly happen)? Is it a system failure? Does the order in which we execute the event logic matter? Can we define a sensible tie-breaking rule?
- Suppose that there are three components rather than two. Is our definition of system state still adequate? What if more than one component can be repaired at a time? Do we need additional events?
- The long-run average number of functional components is not really "system availability." Instead, we want the long-run fraction of time that at least one component is functional. How do we extract this performance measure from Table 1.1? How large should T be to get a good estimate of long-run availability?

Exercises

1. Have each member of the class simulate the TTF example independently until the time of first system failure. Estimate $\mathrm{E}(Y)$, the expected value of the time to first system failure and also a standard error or confidence interval on this estimate. Notice that this is a type of *parallel* simulation since sample paths are being created simultaneously by each class member. Parallel simulation on a computer will be discussed in later chapters.

2. Simulate the TTF example to the time of first system failure with three (one active, two spares) components rather than two.
3. Simulate the TTF example until time $T = 30$ and compute the average number of functional components and average system availability. Unless you are lucky, time 30 will not correspond to a failure or repair event time, so how can you stop the simulation right at time $T = 30$? *Hint:* The system is available if $S(t) > 0$. Average availability is the fraction of time the system is available from $t = 0$ to $t = T$.
4. For the TTF example, suppose that the repair times are not precisely 2.5 days, but instead are 1 day with probability 1/2, and 3 days with probability 1/2. Use a fair coin to generate repair times. Decide on a sensible tie-breaking rule and simulate until the time of the first system failure.
5. If in the TTF example the time to component failure and time to repair a component are exponentially distributed, then the TTF system is a continuous-time Markov chain for which the mean time to first failure and long-run system availability can be derived mathematically. If you know how, do this.
6. Write, in words, the logic for the two system events in the TTF example. Be sure to include updating the value of the state and scheduling any future events.
7. Add to Table 1.1 a column that updates the area $\sum_{i=1}^{j} S(e_{i-1}) \times (e_i - e_{i-1})$ when the jth event executes. Notice that by keeping this running total we can instantly calculate $\bar{S}(e_j)$ at any event time e_j.
8. Add to Table 1.1 a column that updates the area $\sum_{i=1}^{j} I\{S(e_{i-1}) = 2\} \times (e_i - e_{i-1})$ when the jth event executes, where I is the indicator function. Use this to compute the fraction of time that the system is fully functional (no component is in a failed state).
9. Consider a modified TTF system that works as follows. Simulate this system by hand (using dice) until the time of first system failure.
   - There are three components (one active, two spares, but still only one can be repaired at a time); repair time is 3.5 days.
   - In addition, each component is actually made up of two subcomponents, each subcomponent with a time to failure that is equally likely to be $1, 2, \ldots, 6$ days.
   - A component fails when the *first* subcomponent fails. *In other words, to simulate the time to failure of a component, roll the die twice and take the smaller number.*
10. For your simulation in Exercise 9, add a column that updates the area $\sum_{i=1}^{j} S(e_{i-1}) \times (e_i - e_{i-1})$ when the jth event executes, where $S(t)$ is the number of functional components at time t. Use it to calculate the average number of functional components.

Chapter 2
Simulation Programming: Quick Start

As a first step toward more sophisticated simulation programming, this chapter presents a Python simulation of the TTF example in Chap. 1. It also provides a gentle introduction to some important simulation concepts, and a brief overview of Python, leaving the details to Chap. 4. All readers should go through this chapter, even if they will not ultimately program their simulations in Python. The focus here is on basic discrete-event simulation programming principles without using any special simulation support functions or objects. These principles form the core ideas for more sophisticated and scalable programming.

2.1 A TTF Simulation Program

In this section, we show a simple Python implementation of the TTF simulation. Our goal is to demonstrate some key simulation principles that are independent of programming language, and also hint at the more in-depth discussion of Python and PythonSim in later Sects. 4.1 and 4.1.6. For simplicity, this section's implementation does not define user functions or utilize object-oriented programming, but Sect. 4.1.6 provides a (better) version that does both.

We begin by addressing the crucial constituents of a discrete-event simulation. We first discuss these concepts generally, and then apply them to the TTF problem specifically. A discrete-event simulation model represents a system that is characterized by "states." The system state only changes at discrete points in time, and these points are characterized by "events." The simulation advances time on a "clock," moving from event to event in chronological order, and executing a series of actions depending on what type of event just occurred. Event management is usually handled by an "event calendar" that stores upcoming events in chronological order of their occurrence.

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_2

```
# Imports
import random
import math

# Initialize simulation variables for statistics
# S: number of currently functional components (current state)
# SLast: number of functional components at last event
#   (previous state)
# Clock: tracks time in the simulation
# TLast: time of the last event (previous state change)
# Area: sum of states weighted by how much time
#   the simulation was in that state --
#   Area/Clock gives the average number of functional
#   components over the simulation run length
S = 2
SLast = 2
Clock = 0.0
TLast = 0.0
Area = 0.0

# Set random seed
random.seed(1234)
```

Fig. 2.1 Initialization for the TTF problem

The dynamic and flexible nature of discrete-event simulation makes it a powerful and elegant tool for modeling very large and complicated systems with complex logic and many types of events. When using an event-driven simulation approach, we do not need to know or code the exact sequences and occurrences of all events in a simulation run ahead of time. We only need to schedule at least one event up front in each simulation replication, which sets the simulation loop logic in motion. After the initial event, the simulation *dynamically* generates, schedules, and actualizes a sample path of events driven by the stochastic forces in the simulation. From replication to replication, we generally expect these sample paths to be different because each simulation replication has randomness.

When an event happens, three types of actions may occur:

1. The system state is updated;
2. Future events are scheduled; and
3. Statistics are computed or accumulated.

The specific actions generally depend on the type of event that occurs.

Now we explain the TTF simulation and its key concepts. In the TTF simulation, the simulation state is the number of currently functioning components. There are two types of events: failure events and repair events. The state only changes at the occurrence of one of these events. The key variables responsible for the logic in the TTF simulation are `S`, representing the number of functioning components, `Clock`, the current time on the simulation clock, `NextEvent`, which stores the next event's type, NextFailure, which holds the time of the next pending component failure, and NextRepair, the time of the next component repair.

These key variables are initialized in Fig. 2.1. We briefly note that the import statements at the beginning of the code refer to two built-in Python modules that provide additional functions for us to call. The math module allows us to use ∞ and the random module allows us to set a seed for random generation, which we do with the line random.seed(1234). Later on, we discuss these modules and other imports in greater detail in Sect. 4.1. We discuss setting random seeds in Sect. 2.2.

As shown at the beginning of Fig. 2.2, we initialize NextEvent, NextFailure, and NextRepair. We need to schedule at least one event to start the simulation. Since the TTF simulation starts with both components being functional, the only possible next event is a failure, so we randomly generate a time until the next failure and assign this value to NextFailure. Because at this time, a repair event is impossible, we set NextRepair to math.inf. This does not mean that a repair event will never occur in the simulation run, only that at the current simulation time (time 0 since Clock = 0), a repair event is not on the event calendar.

The function random.random() generates a random value that is approximately uniformly distributed between 0 and 1. Multiplying this number by 6 yields a value between 0 and 6; the math.ceiling function rounds this up to an integer. In Sect. 2.2 we explain why this procedure successfully generates a value uniformly at random from $\{1,2,3,4,5,6\}$.

We also note that there is at most one pending failure event and at most one pending repair event. Because of this simplicity, we do not need to code an actual event calendar. For more complicated problems, only keeping track of NextEvent, NextFailure, and NextRepair is insufficient and an event calendar is needed. One can imagine a variant of the TTF problem with 3 components rather than 2 and the ability to repair more than 1 component at a time. In this case, there can be up to two pending repair events, which cannot be captured by our current event management approach.

Since the system fails when there are no more functional components, the simulation loop runs while S > 0. The loop has a clear structure: in each iteration, we determine the next event and advance the clock, then take actions according to the type of event occurring. For example, if NextFailure < NextRepair, then the next event is a failure event, and we update NextEvent accordingly as well as move Clock to this new event time. We set NextFailure to math.inf because we have already incorporated its previous value, and we will randomly generate a new failure time later in the loop. The logic is analogous when the next event is a repair event instead.

Recall that when an event occurs, these actions may follow: the system state is updated, future events are scheduled, and statistics are updated. At a failure event, we update the system state by decreasing the number of functioning components S by 1. If this leaves the system with only one functional component (S = 1), then just prior to this event there were no failed components nor any component under repair. Therefore, we schedule future events: we schedule the failure time of the spare component that has just became active and schedule the repair completion for the active component that has just failed. Events are always scheduled at the current `Clock` time plus an increment into the future. The time to failure of a component is random, but repairs deterministically take 2.5 days, so `NextRepair = Clock + 2.5`. At a repair event, we update the system state by increasing the number of functioning components `S` by 1. If this leaves the system with only one functional component (`S = 1`), then we schedule the next failure and repair event, just as in the `if NextEvent == "Failure"` case.

The last part of each event routine, for both a failure and a repair event, is statistics updating. Recall that we show in Chap. 1 that the time average number of functional components can be expressed as

$$\bar{S}(T) = \frac{1}{T}\int_0^T S(t)\,\mathrm{d}t = \frac{1}{e_N - e_0}\sum_{i=1}^{N} S(e_{i-1}) \times (e_i - e_{i-1}),$$

where e_i are the event times. To avoid having to save all of the event times and values taken by the state variable, a standard trick is to keep a running sum of the area under the curve, updating it each time an event is executed. Specifically,

$$S(e_{i-1}) \times (e_i - e_{i-1}) = \texttt{Slast * (Clock - Tlast)}$$

so that `Clock` is the time of the current event (e_i), `Tlast` is the time of the most recent prior event (e_{i-1}), and `Slast` is the value of $S(t)$ just before the change.

The final section of the main program reports two output results: the time of system failure and the average number of functional components from time 0 until the system first fails. Notice that since the simulation ends as soon as `S = 0`, the simulation `Clock` is also the time of system failure. And since we have been accumulating the area under the $S(t)$ curve in the variable `Area`, `Area/Clock` is the average value of $S(t)$ up to that time.

Running Figs. 2.1 and 2.2 in Python produces 1.65 for `Area/Clock` (average number of functional components) and 10 for `Clock` (time of system failure). We note that this code only simulates 1 replication. We demonstrate how to run multiple replications in Sect. 2.4.

2.2 Random-Variate Generation

In this section, we discuss how to generate random variates. A stochastic simulation is driven by fully characterized probability models which generate realizations. We use the term "realization" to mean specific, usually numerical, outcomes, and we call the process of creating these realizations *random-variate generation*. The most basic probability models we need are i.i.d. random variables $X_1, X_2, \ldots$ with known distribution F_X. Because our simulations are built from random variables, our anal-

```
# Initialize first event
NextEvent = ""
NextFailure = math.ceil(6*random.random())
NextRepair = math.inf

# Main simulation loop
while S > 0:

    # Determine what type of event is next
    #   and advance time to this event time
    if NextFailure < NextRepair:
        NextEvent = "Failure"
        Clock = NextFailure
        NextFailure = math.inf
    elif NextFailure >= NextRepair:
        NextEvent = "Repair"
        Clock = NextRepair
        NextRepair = math.inf

    # Update S and upcoming events based
    #   on event type
    # If next event is a failure,
    #   number of functional components decreases by 1.
    #   If there is 1 functional component,
    #       schedule the next failure and repair times.
    if NextEvent == "Failure":
        S = S - 1
        if S == 1:
            NextFailure = Clock + math.ceil(6*random.random())
            NextRepair = Clock + 2.5
    # If next event is a repair,
    #   number of functional components increases by 1.
    elif NextEvent == "Repair":
        S = S + 1
        if S == 1:
            NextFailure = Clock + math.ceil(6*random.random())
            NextRepair = Clock + 2.5

    # Update variables for computing the
    #   time-average number of functional components.
    Area = Area + SLast * (Clock - TLast)
    TLast = Clock
    SLast = S

print(Area/Clock)
print(Clock)
```

Fig. 2.2 Main simulation loop and results printing for the TTF problem

ysis is ultimately statistical rather than mathematical, and we *estimate* performance measures of interest from simulated data.

Suppose we have an algorithm called `algorithm`, which takes a single numerical input u and produces a single numerical output x; that is

$$x = \texttt{algorithm}(u).$$

Here `algorithm` is typically some computer code, but the important point is that it is a deterministic input–output transformation. For instance, if $\texttt{algorithm}(0.3)$ returns the value $x = 2$, then $\texttt{algorithm}(0.3)$ always returns the same value of 2 each time it is called with 0.3 as its argument.

Suppose that the input u is not a specified constant, but instead is a random variable, say U, with a known probability distribution F_U. Then `algorithm` defines a random variable

$$X = \texttt{algorithm}(U)$$

and it makes perfect sense to talk about probabilistic quantities such as $\Pr\{X \le 20\}$, $\mathrm{E}(X)$, or the distribution F_X. The important concept here is that `algorithm`, which is a recipe that takes a numerical input and yields a numerical output, can also be thought of as a way to transform a random variable U with some distribution into a random variable X with a different distribution.

Now that we have introduced the concept of `algorithm` with random input U, we introduce the very important case in which F_U is the uniform distribution on the interval $[0,1]$. This distribution is often denoted $U(0,1)$ and is defined as

$$F_U(u) = \Pr\{U \le u\} = \begin{cases} 0, \ u < 0 \\ u, \ 0 \le u < 1 \\ 1, \ 1 \le u. \end{cases} \tag{2.1}$$

$U(0,1)$ random variables are fundamental elements in random-variate generation, as demonstrated in the example below.

Example 2.1. We show how to use a uniform random variable to generate a realization of an exponential random variable with mean 1. Suppose that $\texttt{algorithm(u)} = -\ln(1-u)$, where ln is the natural logarithm. Let $U \sim U(0,1)$. Then we have

$$\begin{aligned} \Pr\{\texttt{algorithm}(U) \le x\} &= \Pr\{-\ln(1-U) \le x\} \\ &= \Pr\{1 - U \ge \mathrm{e}^{-x}\} \\ &= \Pr\{U \le 1 - \mathrm{e}^{-x}\} \\ &= 1 - \mathrm{e}^{-x}, \ x \ge 0. \end{aligned} \tag{2.2}$$

The last step follows because $\Pr\{U \le u\} = u$ for $0 \le u \le 1$. Expression (2.2) is the cumulative distribution function (cdf) of the exponential distribution with mean 1, showing that we have an algorithmic description of this random variable. Thus, if we had a way to produce realizations of U, we could then generate realizations of the exponential with mean 1; this is random-variate generation.

Next we show how to *construct* `algorithm` so that X has a distribution F_X that we desire, rather than wondering what F_X we get when we apply some `algorithm` to U. Being able to do this is essential to stochastic simulation.

To start off, consider the case in which the cdf F_X is a continuous, increasing function for all x such that $0 < F_X(x) < 1$; this is true for the exponential, normal, lognormal, Weibull, uniform, and many other well-known distributions. Under these conditions the equation $u = F_X(x)$ has a unique solution which we denote by $x = F_X^{-1}(u)$, the *inverse cdf*. Then for $U \sim U(0,1)$, define the random variable X by $X = F_X^{-1}(U)$. We can immediately show that

$$\begin{aligned}\Pr\{X \le x\} &= \Pr\{F_X^{-1}(U) \le x\}\\ &= \Pr\{U \le F_X(x)\}\\ &= F_X(x).\end{aligned}$$

This is the same logic behind the derivation in Expression (2.2) for the exponential distribution with mean 1 for which $F_X^{-1}(u) = -\ln(1-u)$.

Even when F_X^{-1} is not available in closed form, as it is for the exponential, the cdf can be inverted numerically using a root-finding algorithm. Thus, the inverse cdf method is a general-purpose method for distributions with continuous, increasing cdfs.

Now consider the more general case in which we have a random variable with a cdf F_X which is not necessarily continuous or increasing. We can define the inverse generally to work for any random variable:

$$F_X^{-1}(u) = \min\{x : F_X(x) \ge u\}. \tag{2.3}$$

This formula handles the case in which F_X is continuous and increasing, but also works more broadly, for example, for cdfs which contain jumps. Using Equation (2.3), we can generate any random variable X with cdf F_X by using $X = F_X^{-1}(U)$, where $U \sim U(0,1)$.

In the following examples, we apply our inverse cdf technique to discrete distributions. In Chap. 3 we introduce examples requiring other probability models, including the exponential distribution.

Example 2.2. Consider random-variate generation for a discrete distribution. Suppose that random variable X can take values $x_1 < x_2 < x_3 < \cdots$ with corresponding probabilities $p_1, p_2, p_3, \ldots$, where $\sum_i p_i = 1$. For the time to component failure in the TTF system, we have $x_i = i$ and $p_i = 1/6$ for $i = 1,2,\ldots,6$. We apply Definition (2.3) and verify that

$$\begin{aligned}\Pr\{X = x_j\} &= \Pr\{F_X^{-1}(U) = x_j\}\\ &= \Pr\{F_X(x_j) \ge U > F_X(x_{j-1})\}\\ &= \Pr\{F_X(x_{j-1}) < U \le F_X(x_j)\} \qquad (2.4)\end{aligned}$$

$$\begin{aligned}
&= \Pr\{U \le F_X(x_j)\} - \Pr\{U \le F_X(x_{j-1})\} \\
&= F_X(x_j) - F_X(x_{j-1}) \\
&= \sum_{i=1}^{j} p_i - \sum_{i=1}^{j-1} p_i \\
&= p_j
\end{aligned} \tag{2.5}$$

as desired. Expression (2.4) provides the basic algorithm for discrete distributions: given U, find the smallest j such that $\sum_{i=1}^{j} p_i \ge U$ and return x_j.

Example 2.3. In the TTF simulation (see Fig. 2.2), we need to generate component failure times that are i.i.d. with a discrete uniform distribution on $\{1,2,\ldots,6\}$. The possible realizations are $x_i = i$ with respective probabilities $p_i = 1/6$ for $i = 1,2,\ldots,6$. For a general discrete uniform distribution on $\{1,2,\ldots,k\}$, we have $x_i = i$ and $p_i = 1/k$ for $i = 1,2,\ldots,k$. The TTF component failure times come from a specific case of $k = 6$.

Expression (2.4) implies we need to find j such that

$$\frac{j-1}{k} < U \le \frac{j}{k}. \tag{2.6}$$

The end-of-chapter exercises include showing that this equation leads to the formula $X = \lceil Uk \rceil$, where $\lceil \cdot \rceil$ is the ceiling function that returns the smallest integer that is greater than or equal to its argument.

> We say that a random-variate generation `algorithm` is <u>exact</u> if, under the assumption of $U(0,1)$ variates as input, and infinite precision for the numerical variables and functions that make up `algorithm`, `algorithm` implies the desired distribution using an argument such as (2.2).

Our algorithms for the exponential distribution and discrete uniform distribution in Examples (2.2–2.3) are exact.

In practice there are compromises. First, computer implementation of functions such as ln do not work perfectly, if for no other reason than the computer representation of numbers is finite. This is an issue in all numerical programming. The second compromise is more central to simulation and has to do with generating realizations of U, and this is the next topic of discussion.

2.3 Random-Number Generation

Producing realizations of U is called *random-number generation*. The definition of exact random-variate generation for X requires that $U \sim U(0,1)$; therefore, an exact method to generate $X_1, X_2, \ldots$ which are i.i.d. F_X requires $U_1, U_2, \ldots$ which are i.i.d. $U(0,1)$. Because of the finite representation of numbers in a computer, this is not strictly possible. Further, even if possible, it is still a bad idea, since it leads to simulation results that are not reproducible. A replication of a simulation using truly random numbers produces different output each time, even if nothing in the simulation has changed.

Therefore, instead of using truly random numbers, stochastic simulations use *pseudorandom numbers*. Consider a large, ordered list of numbers between 0 and 1:

$$U_1, U_2, U_3, \ldots, U_i, U_{i+1}, \ldots, U_{P-1}, U_P, U_1, U_2, \ldots.$$

The list needs to be quite long (about two billion in older simulation programs, much, much longer in newer ones). There may be repeats in this list, but when the end of the list is reached it starts over from the beginning; for this reason P is called the *period* of the pseudorandom numbers. A good list of pseudorandom numbers appears to be statistically indistinguishable from random numbers, provided that the count of random numbers that a simulation uses is significantly less than P.

Pseudorandom numbers are not random in any sense—they are a predetermined, finite list that eventually repeats. Consider the following Python code snippet

```
import random
random.seed(1234)
random.random()
```

We import the module `random`, which implements pseudorandom-number generation. The line `random.seed(1234)` seeds the starting point in the pseudorandom-number list, and the function `random.random()` draws pseudorandom numbers out in order from that starting point. This code snippet returns the same result each time it is run. Similarly, because we set `random.seed(1234)` at the beginning of the TTF simulation (see Fig. 2.1), running the TTF simulation (see Fig. 2.2) also returns the same result each time it is run. If `1234` is replaced by a different integer, then the pseudorandom numbers are drawn starting from a different place in the list and thus the result would be different (we invite the reader to test this fact for themselves).

Storing an actual list of pseudorandom numbers would be cumbersome at best, and impossible if P is large enough. Therefore, the pseudorandom numbers in simulation software are produced by a recursive algorithm; we describe the construction of such pseudorandom-number generators in Chap. 6, but the list representation is conceptually correct. Using a pseudorandom-number generator, rather than random numbers, is an important practical compromise. The full implications of this compromise are discussed throughout the book.

2.4 Replications

As discussed in Chap. 1, one justification for using simulation-based estimators is that as the number of replications increases, the sample average across replications converges (with probability 1) to the true long-run average. In this section we introduce the key ideas behind programming replications.

To be concrete, once again consider the TTF example. Let Y be a random variable representing time to system failure and let $\bar{S}(Y)$ be the average number of functional components up to time Y. Suppose we want to simulate 100 i.i.d. replications, save the results of Y and $\bar{S}(Y)$ from each replication, and report each statistic's overall average across the replications. Specifically, we want

$$\bar{Y} = \frac{1}{100}\sum_{i=1}^{100} Y_i$$
$$\bar{S} = \frac{1}{100}\sum_{i=1}^{100} \bar{S}_i(Y_i),$$

where the subscript i represents the results from the ith replication.

To achieve good estimation, we want each Y_i and $\bar{S}_i$ to be i.i.d. for $i = 1,2,\ldots,100$. Identically distributed observations require that the simulation logic, including the initial conditions, are the same on each replication. The assumption of independent observations means that the component failure times on any replication are statistically independent of the failure times on any other replication.

We can achieve (approximately) i.i.d. replications and thus i.i.d. Y_i and $\bar{S}_i$ provided that we *do not* reset the pseudorandom-number generator between replications, but *do* reset everything else. This is because the sequence of random numbers that dictate failure times are produced by a pseudorandom-number generator approximating i.i.d. behavior, provided we just let the generator keep running between replications and do not start over.

Figure 2.3 shows a modification of the TTF simulation's main program to implement replications. The initialization part is the same as in Fig. 2.1. Notice that the code within the `while S > 0` loop is unchanged from Fig. 2.2. Since that code section is unchanged, code comments are removed for readability. The "while" loop is nested within a "for" loop that cycles through `NumReps` replications, where we set `NumReps = 100`.

The pseudorandom-number generator is initialized once outside the replication loop, in Fig. 2.1. The simulation variables `S`, `SLast`, `Clock`, `TLast`, `Area` are reinitialized inside the replication loop, meaning that they are reset at the beginning of each replication. The variables `S` and `SLast` are both reset to 2 since each replication commences with 2 functional components, `Clock` resets to 0, and `TLast` and `Area`, which are used to compute the average number of functional components in a replication, start over at 0. The initial failure and repair events are reset inside the replication loop as well. In this way, we achieve (approximately) i.i.d. results.

A key to simulation experiment design is choosing the number of replications large enough so that where we start the pseudorandom-number generator does not matter.

Notice how at the end of each replication we update sums of the average number of functional components and the time of system failure, so that when all of the replications are complete, we can easily compute the across-replication aver-

```
# Specify number of replications
NumReps = 100

# Initialize additional simulation variables for
#   across-replication statistics
# SumS: sum of average number of functional components
#   in each replication
# SumY: sum of time-to-failure in each replication
SumS = 0.0
SumY = 0.0

# Main simulation loop
for rep in range(NumReps):
    NextEvent = ""
    NextFailure = math.ceil(6*random.random())
    NextRepair = math.inf
    S = 2
    SLast = 2
    Clock = 0.0
    TLast = 0.0
    Area = 0.0

    while S > 0:
        if NextFailure < NextRepair:
            NextEvent = "Failure"
            Clock = NextFailure
            NextFailure = math.inf
        elif NextFailure >= NextRepair:
            NextEvent = "Repair"
            Clock = NextRepair
            NextRepair = math.inf

        if NextEvent == "Failure":
            S = S - 1
            if S == 1:
                NextFailure = Clock + math.ceil(6*random.random())
                NextRepair = Clock + 2.5
        elif NextEvent == "Repair":
            S = S + 1
            if S == 1:
                NextFailure = Clock + math.ceil(6*random.random())
                NextRepair = Clock + 2.5

        Area = Area + SLast * (Clock - TLast)
        TLast = Clock
        SLast = S

    SumS += Area / Clock
    SumY += Clock

print(SumS / NumReps)
print(SumY / NumReps)
```

Fig. 2.3 Main simulation loop and results printing for the TTF problem with replications

ages. Updating the variables `SumS` and `SumY` occurs outside of the "while" loop but within the replication loop, after the simulation run finishes, as shown in Fig. 2.3.

Tracking cumulative sums is an alternative to saving each replication's results and computing the overall average later. The cumulative sum approach has the advantage of saving memory. On the other hand, if each replication's results are not saved, it may be impossible to compute new statistics afterwards, if the necessary data for these statistics are not available.

Running the code in Figs. 2.1 and 2.3 gives an average of 1.57 functional components and an average of 13.3 days until system failure, based on 100 replications.

Exercises

1. Modify the TTF simulation so that it can have any number of components, not just 2. Assume that components can still be repaired only one at a time.
2. Simulate the TTF example until time $T = 1000$ and report the average number of functional components. Create a variable `NextEndSimulation` and set its value to 1000. Then the possible types of events are `NextFailure`, `NextRepair` and `NextEndSimulation`. Code the additional logic necessary to get the simulation to stop at time 1000. Why is this a more effective approach than changing the loop condition to `while S > 0 and Clock < NextEndSimulation`?
3. For the TTF simulation, suppose that the repair times are not precisely 2.5 days, but instead are 1 day with probability 1/2, and 3 days with probability 1/2. Decide on a sensible tie-breaking rule, modify the simulation and simulate until the time of the first system failure.
4. The TTF system is available provided there is at least one functional component. Define a variable $A(t)$ that takes the value 1 when the system is available, and is 0 otherwise. Then

$$\bar{A}(T) = \frac{1}{T}\int_0^T A(t)\,\mathrm{d}t$$

   is the average system availability from time 0 to T. Modify the simulation created for Exercise 2 to also report average system availability.
5. For each of the exercises above, add the capability to do replications. Using 100 replications, estimate the expected value of each output and a 95% confidence interval on it.
6. Show that (2.6) implies that $X = \lceil Uk \rceil$ is the inverse cdf for the discrete uniform distribution on $\{1, 2, \ldots, k\}$.
7. An alternative to using $X = \lceil Uk \rceil$ to generate variates from a discrete uniform distribution on $\{1, 2, \ldots, k\}$ is $X = \lfloor Uk + 1 \rfloor$. Both methods have a flaw if it is possible for $U = 0$ or $U = 1$. What is it? For this reason (and others) pseudorandom-number generators are designed so that they do not generate exactly 0 or 1.

8. Derive a method for generating a discrete uniform distribution on $\{a, a+1, a+2, \ldots, b\}$ where a and b are integers with $a < b$.
9. Modify the Python code for the TTF simulation with replications to match the description from Exercise 9 of Chap. 1 to obtain the average number of functional components and the time to failure from each replication. Using 100 replications, estimate the expected value of each performance measure and a 95% confidence interval on it.
10. A very simple inventory system works as follows. At time 0 it has 50 items in stock. Each day there is a demand for items which is equally likely to be $1, 2, \ldots, 9$ items. The system satisfies as much of the demand as possible, but if it does not have enough items in stock then the excess demand is lost. When the stock reaches or goes below `ReorderPoint`, an order is placed for 50 additional items; it takes $1, 2$, or 3 days for the order to arrive, equally likely. There can only be one pending order at a time, meaning that if an order has been placed and has not yet arrived then another order cannot be placed. Orders arrive before demands each day, and the planning horizon is 365 days. Build a simulation model of this system, and use it to find the smallest value of `ReorderPoint` that guarantees that there are no items in stock less than 5% of the time. Also estimate the mean stock level for your policy. Put a 95% confidence interval around your estimate. Start with 100 replications, but increase this as necessary to be sure of your recommendation.

Chapter 3
Examples

None of the examples in this chapter need to be simulated; they can all be analyzed by mathematical/numerical analysis. But thinking about how to simulate them will help us understand what works, and why, for systems we do need to simulate. These examples also provide a test bed for evaluating new ideas in simulation design and analysis. You may have encountered these models in other classes, but it is neither necessary to have seen them before, nor to know how the results are derived, to use them throughout the book.

3.1 $M(t)/M/\infty$ Queue

This example is based on Nelson (1995, Chap. 8).

Example 3.1 (The Parking Lot). Prior to building a large shopping mall, the mall designers must decide how large a parking garage is needed. The arrival rate of cars will undoubtedly vary over time of day, and even day of year. Some patrons will visit the mall for a brief time, while others may stay all day. Once the garage is built, the mall may open and close floors depending on the load, but the first question is, what should the maximum capacity of the garage be?

Since the mall developers would like virtually everyone to be able to have a parking space, a standard modeling trick is to pretend that the parking garage is infinitely large and then evaluate the probability distribution of the number of spaces actually used. If the capacity of the garage is set to a level that would rarely be exceeded in an infinitely large garage, then that should be adequate in practice.

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_3

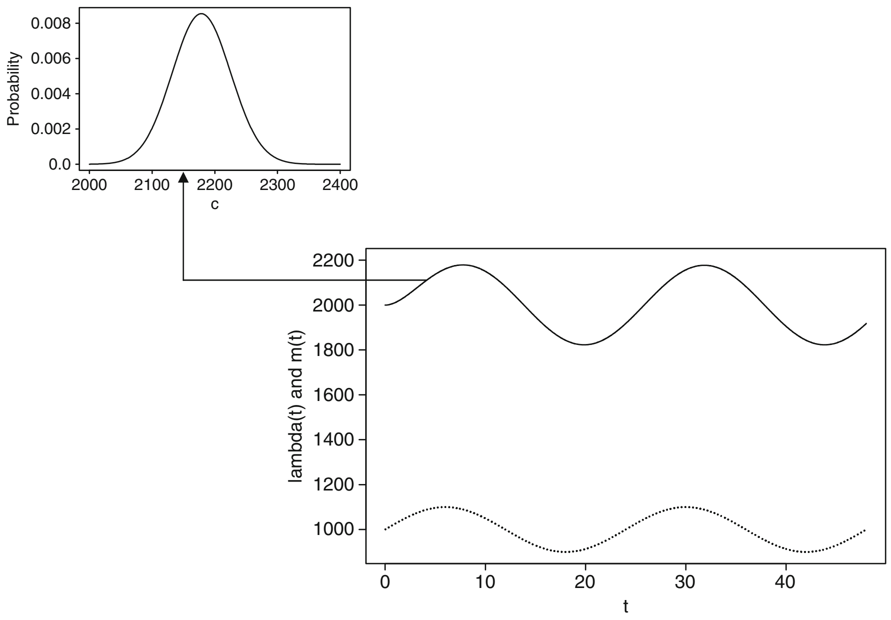

Fig. 3.1 Plot of $\lambda(t) = 1000 + 100\sin(\pi t/12)$ (*bottom*) and $m(t)$ (*top*) when $\tau = 2$, and the Poisson distribution of number of cars in the garage with mean $m^{\star}$ (*top left*)

The $M(t)/M/\infty$ queue is a service system in which customers arrive according to a nonstationary Poisson arrival process with time-varying arrival rate $\lambda(t)$ customers/time,[1] the service time is exponentially distributed with mean τ, and there are an infinite number of servers. In Example 3.1 the customers are cars arriving at a rate of $\lambda(t)$ per hour, a "service time" is the time spent occupying a space with mean τ hours, and we pretend that there are infinitely many spaces. A good reference for using queueing models to address the design of service systems like call centers or the parking lot is Whitt (2007).

Let $N(t)$ be the number of cars in the parking garage at time $t \geq 0$. Then it is known that $N(t)$ has a Poisson distribution with mean $m(t)$, where $m(t)$ solves the differential equation

$$\frac{d}{dt}m(t) = \lambda(t) - \frac{m(t)}{\tau} \tag{3.1}$$

with an appropriate initial condition, say, $m(0) = 0$ if the garage starts empty (see Nelson and Taaffe, 2004). One way to set the capacity of the parking garage is to solve (or numerically integrate over time) Eq. (3.1) to find the largest value attained

[1] A stationary Poisson arrival process has independent times between arrivals that are exponentially distributed with mean $1/\lambda$ or equivalently arrival rate λ. The nonstationary Poisson process is a generalization that allows the arrival rate to change over time; it is covered in detail in Chap. 6.

by $m(t)$, say $m^\star$. Then at its most congested, the distribution of the number of cars in the garage is Poisson with mean $m^\star$, which implies that

$$\Pr\{\text{number of cars in the garage } \le c\} = \sum_{n=0}^{c} \frac{e^{-m^\star}(m^\star)^n}{n!}. \tag{3.2}$$

Thus, c can be adjusted until this probability is sufficiently close to 1 (e.g., 0.99). Figure 3.1 shows a plot of the arrival rate to the garage, $\lambda(t)$, the mean number of cars in the garage, $m(t)$, and the probability distribution of the number of cars in the garage at time $t^\star$ when the mean number is $m^\star$. (*Note*: The Poisson is a discrete distribution, but is plotted as a continuous curve here.)

Although there is no need to simulate an $M(t)/M/\infty$ queue to design the garage, only slightly enriching the model would make it mathematically intractable. For instance, in a large garage it might be important to account for drivers who have difficulty finding a space, or to include drivers who occupy multiple spaces to protect their cars and employees who also park in the garage but stay 8 h. These changes and others could necessitate simulation; however, the behavior would be similar to the $M(t)/M/\infty$ results.

If we needed to solve the parking lot problem via simulation, what issues would arise that we did not see in the TTF problem?

- Like the TTF problem, the system state will include (at least) the number of cars currently in the garage, and the events will include car arrivals and departures, similar to the failure and repair events in the TTF simulation. However, a single "Next Departure" event will not be sufficient;[2] in fact, there will need to be one pending departure event for each car currently in the garage. Effective ways to program simulations that do not have a small, fixed number of events are covered in Chap. 4.
- To build the simulation we need the facility to model and simulate a time-varying, stochastic arrival process and random parking times. Chapter 2 hinted at how we generate random variates, but what is needed here is clearly another level of sophistication. Methods for fitting and simulating input processes are covered in Chap. 6.
- We need to track $N(t)$, the number of cars in the garage, over some relevant time horizon. Unlike the average system availability in the TTF problem, the average number of cars in the infinite-sized garage is not relevant unless we are satisfied with having many cars turned away. The maximum observed value of $N(t)$ might seem better, but in a simulation we could, by chance, observe a one-in-a-million event; do we want to size the garage to protect against something that rare? How can we estimate something like (3.2) using simulation? These issues are covered in Chap. 7.

[2] In the special case of exponentially distributed parking times one departure event is adequate due to the memorylessness property of the distribution; however, this will not be the case generally.

3.2 *M/G/*1 Queue

Example 3.2 (Hospital Reception). The receptionist in a medium-sized hospital helps direct entering patients and visitors to the relevant floor or wing of the building. Under discussion is replacing the human receptionist with an electronic kiosk with a touch screen that might be a bit slower, or more variable, for patients and visitors who are less comfortable interacting with a touch screen. The hospital management engineers want to evaluate how much additional delay that this might cause.

For a rough-cut analysis of the situation the management engineers could use hospital records to estimate an overall arrival rate of patients and visitors to the reception desk and collect data on people interacting with the kiosk from a trial study with the vendor. Pretending that the characteristics of any system are unchanging over time is always an approximation, but such approximations often lead to useful and mathematically tractable models. Such is the case for the $M/G/1$ queue, a single-server queueing system to which customers arrive according to a Poisson process with rate λ customers/time, and whose service times are i.i.d. random variables with mean τ and standard deviation σ time units. The $M/G/1$ model is mathematically tractable, and illustrates several important ideas for computer simulation.

Let $A_1, A_2, \ldots$ be a sequence of i.i.d. random variables with mean $1/\lambda$ representing the interarrival times between customers (where A_1 is the actual time of the first arrival). Similarly, let $X_1, X_2, \ldots$ be the i.i.d. service times of successive customers with mean τ and standard deviation σ. Then if $Y_1, Y_2, \ldots$ are the successive waiting times in queue (i.e., the time from customer arrival until service begins), a little thought reveals that

$$Y_i = \max\{0, Y_{i-1} + X_{i-1} - A_i\},\ i = 1, 2, \ldots, \tag{3.3}$$

where we need to define $Y_0 = X_0 = 0$ to make the recursion work. This is known as Lindley's equation (Gross et al., 2008, p. 14), which not only is a convenient way to simulate the $M/G/1$ waiting-time process, but also provides a number of insights about many simulation output processes:[3]

- The successive waiting times are dependent, since Y_i clearly depends on Y_{i-1} (if the customer in front of me waited a long time, I probably will too, unless I arrive well after her).
- The outputs are not identically distributed. Clearly $Y_1 = 0$ (the first customer arrives to an empty system and does not wait), but the other Y_i's are not certain to be 0.

[3] Lindley's equation makes queueing simulation look deceptively easy. However, if there are ten servers instead of one, or if customers are served in some priority order, or if we are interested in the number of customers in the queue rather than the waiting time, then an event-based simulation is typically required. Exercise 4 asks you to develop such a simulation; see also Chap. 4.

- No matter how many customers we simulate ($i \to \infty$), the waiting times remain random variables.

This sounds like a difficult system to analyze, since the data are neither independent nor identically distributed, and they do not converge to some constant that we might use to summarize the system. What we might hope, however, is that there is a more complicated, but still useful, limiting behavior.

Provided $\lambda < \infty$ (customers do not arrive infinitely fast), $\rho = \lambda\tau < 1$ (on average we can serve customers at least a little faster than they arrive), and $\sigma < \infty$, then a number of things happen as $i \to \infty$:

1. The outputs $Y_1, Y_2, \ldots$ converge in distribution[4] to a random variable Y whose distribution is not a function of the customer number i. That is, even though customer waiting times are always random variables, the distribution of those random variables is no longer changing. This makes the distribution of Y a nice summary of long-run performance, but it is not clear how to simulate it.
2. The sample mean $\overline{Y}(m) = m^{-1}\sum_{i=1}^{m} Y_i$ converges with probability 1 to a constant μ which we can interpret as the long-run average waiting time. This is less descriptive than the distribution of Y, but is still a useful summary measure. More importantly, it is easy to see how to estimate μ via simulation: Take the average of a very large number of simulated waiting times generated by using (3.3).
3. The $\mathrm{E}(Y)$ and μ are equal, and are given by the Pollaczek–Khinchine formula (Gross et al., 2008, Sect. 5.1.1)
$$\mu = \mathrm{E}(Y) = \frac{\lambda(\sigma^2 + \tau^2)}{2(1 - \lambda\tau)}. \tag{3.4}$$
Using this formula the management engineers can evaluate the impact of an increased mean service time (τ) and increased variability (σ^2) without the need to simulate. Unfortunately, as in the parking lot example, relatively small enhancements of the model to make it more realistic also make it mathematically intractable. However, (3.4) still provides insight about how arrival rates, mean service times, and service variability affect queueing performance: As $\rho = \lambda\tau$ approaches 1 the mean waiting time increases explosively, while it increases linearly in the variance of the service times.

Suppose we used the natural estimator $\overline{Y}(m)$ to estimate μ via simulation. Obviously we have to choose the number of patients and visitors to simulate $m < \infty$ because we have to stop the simulation to obtain our estimate. How many patients and visitors are enough? One way to answer that question is by picking m large enough so that the standard error of $\overline{Y}(m)$, which is $\sqrt{\mathrm{Var}\left(\overline{Y}(m)\right)}$, is small enough. The standard error can be interpreted as the "average error" in $\overline{Y}(m)$ and often appears in confidence intervals.

[4] The various modes of convergence (in distribution, with probability 1, and others) are defined in Chap. 5.

In Chap. 5 we will discuss the fact that for certain types of simulation output processes the limit

$$\gamma^2 = \lim_{m\to\infty} m\mathrm{Var}\left(\overline{Y}(m)\right) \tag{3.5}$$

exists, even though the output data $Y_1, Y_2, \ldots$ are not i.i.d. (it is easy to show that this limit exists when the data are i.i.d.). We call γ^2 the *asymptotic variance*, and it suggests that for m large enough, $\sqrt{\mathrm{Var}\left(\overline{Y}(m)\right)} \approx \gamma/\sqrt{m}$.

In the special case of an $M/M/1$ queue (which means the service time is exponentially distributed) and with $\tau = 1$, it can be shown that

$$\gamma^2 = \frac{\rho(2+5\rho-4\rho^2+\rho^3)}{(1-\rho)^4} \approx \frac{4\rho}{(1-\rho)^4}$$

(Whitt, 1989), so that

$$\sqrt{\mathrm{Var}\left(\overline{Y}(m)\right)} \approx \frac{2\sqrt{\rho}/(1-\rho)^2}{\sqrt{m}}. \tag{3.6}$$

Notice that how much simulation effort is required depends dramatically on how close the system is to its capacity (i.e., how close ρ is to 1): For instance, m needs to be about 1000 times larger to obtain the same standard error at $\rho = 0.9$ as at $\rho = 0.5$ (verify this for yourself using Eq. (3.6)). This is a feature of simulations that involve queues or networks of queues. Determination of the run length m, and estimation of γ^2, are topics of Chap. 8.

3.3 The AR(1) Surrogate Model

The examples in this chapter are simplified, and therefore tractable, versions of realistic models that we might actually simulate. The lone exception is the AR(1) model presented here.

AR(1) is an abbreviation for "autoregressive order-1"; it is a model that arises naturally in time series forecasting (e.g., Chatfield, 2004). Here we use it as a stand-in, or surrogate, for output from a stochastic simulation experiment because it shares the properties of many simulation output processes, but is very readily analyzed.

The AR(1) model for the ith simulation output (within a single replication) is

$$Y_i = \mu + \varphi(Y_{i-1} - \mu) + X_i,\ i = 1, 2, \ldots \tag{3.7}$$

with the following conditions:

- $X_1, X_2, \ldots$ are i.i.d. random variables with mean 0 and finite variance σ^2; thus, the X's represent the stochastic nature of the simulation output.
- $|\varphi| < 1$; thus, φ controls the strength of dependence among successive outputs, with $\varphi = 0$ corresponding to i.i.d. output, $0 < \varphi < 1$ corresponding to positive dependence, and $-1 < \varphi < 0$ corresponding to negative dependence.

- μ is a constant; thus, μ is the long-run average, which is often the performance measure of interest.
- The distribution of Y_0 is given (including the typical case where it is a constant) and independent of the X's; thus, Y_0 represents the initial condition at the beginning of the simulation which clearly has an impact on the outputs that follow.

The similarity of the AR(1) model to Lindley's equation (3.3) for the M/G/1 queue

$$Y_i = \max\{0, Y_{i-1} + X_{i-1} - A_i\},\ i = 1, 2, \ldots$$

is apparent, but the absence of the max operator makes the AR(1) easier to analyze. In particular, we can use induction to show that

$$Y_i = \mu + \varphi^i (Y_0 - \mu) + \sum_{j=0}^{i-1} \varphi^j X_{i-j}. \tag{3.8}$$

From Eq. (3.8) we can derive several properties of AR(1) output immediately:

$$\mathrm{E}(Y_i) = \mu + \varphi^i (\mathrm{E}(Y_0) - \mu) \overset{i\to\infty}{\longrightarrow} \mu \tag{3.9}$$

$$\mathrm{Var}(Y_i) = \varphi^{2i}\mathrm{Var}(Y_0) + \sigma^2 \sum_{j=0}^{i-1} \varphi^{2j} \overset{i\to\infty}{\longrightarrow} \frac{\sigma^2}{1-\varphi^2}. \tag{3.10}$$

It can also be shown that $\mathrm{Corr}(Y_i, Y_{i+j}) \overset{i\to\infty}{\to} \varphi^j$, implying that the correlation between outputs is geometrically decreasing with the number of observations (called the "lag") between them.

Now suppose that the X's are normally distributed, and Y_0 is a constant. Then since Y_i is the sum of normally distributed random variables by (3.8), it is also normally distributed. And since a normally distributed random variable is completely characterized by its mean and variance, the limits in (3.9) and (3.10) show that as i increases (the length of the replication becomes longer) the distribution of the AR(1) output becomes independent of i, and specifically converges to a random variable $Y \sim \mathrm{N}\left(\mu, \sigma^2/(1-\varphi^2)\right)$. This illustrates the concept of convergence in distribution, introduced for the M/G/1 queue in Sect. 3.2.

As compared to Lindley's equation, the AR(1) model is particularly useful for mathematically evaluating the features of simulation output that impact output analysis because the key factors that might be important can be individually controlled: variability through σ^2, dependence through φ, and initialization through Y_0.

3.4 A Stochastic Activity Network

This example is based on Burt and Garman (1971) and Henderson and Nelson (2006, Chap. 1, Sect. 2.2).

Example 3.3 (Construction Project). A construction project consists of a large number of activities. Some of these can be completed in parallel (drywall can be ordered while the foundation is being poured), while some cannot commence until others are completed (the roof cannot be constructed until the framing is finished). Since the durations of the activities are not precisely predictable, the project planners would like to take into account this variability when bidding the project because there will be penalties for completing the project after the contract date.

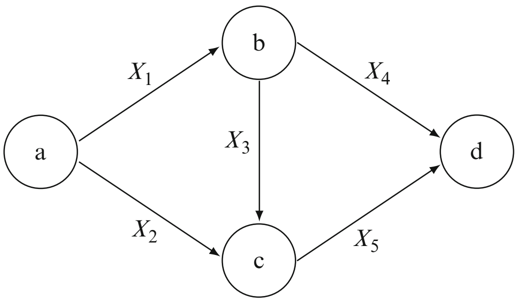

Fig. 3.2 A small stochastic activity network

Project planning problems such as Example 3.3 can sometimes be modeled as stochastic activity networks (SANs); a small instance—which will be used for illustration here—is shown in Fig. 3.2. The nodes (circles) represent project milestones, and the arcs (arrows) are activities. The project starts with all activities emanating from the source (first) node and is completed when the sink (last) node is reached. The rule is that all outgoing activities from a node commence when all of the incoming activities to that node are completed. The duration of the ℓth activity is a random variable X_ℓ. Thus, the time to complete the project will be the longest path through the network:

$$Y = \max\{X_1 + X_4, X_1 + X_3 + X_5, X_2 + X_5\}. \tag{3.11}$$

The project planners are interested in information about the distribution of Y, for instance, $\theta = \Pr\{Y > t_p\}$, where t_p is the quoted duration of the project.

Now if it happens that the activity durations are independent, and have common exponential distribution with mean 1, then a sequence of careful conditioning arguments gives (Burt & Garman, 1971)

$$\Pr\{Y \le t_p\} = \left(\frac{1}{2}t_p^2 - 3t_p - 3\right) e^{-2t_p} + \left(-\frac{1}{2}t_p^2 - 3t_p + 3\right) e^{-t_p} + 1 - e^{-3t_p}. \tag{3.12}$$

However, the problem quickly becomes intractable if some or all of the distributions are not exponential, or if $X_1, X_2, \ldots, X_5$ are not independent. Then simulation is required. The following simulation could be used to estimate θ:

```
1. set $s = 0$
2. repeat $n$ times:

   a. generate $X_1, X_2, \ldots, X_5$
   b. set $Y = \max\{X_1 + X_4, X_1 + X_3 + X_5, X_2 + X_5\}$
   c. if $Y > t_p$ then set $s = s + 1$

3. estimate $\theta$ by $\widehat{\theta} = s/n$
```

Let $Y_1, Y_2, \ldots, Y_n$ be the n project completion times generated by this simulation. Assuming the X's are generated independently on each trial, then the Y's are naturally i.i.d. and classical statistical analysis applies.

Like some earlier examples, an event-based simulation is not required for this small problem; however, in a project with a realistic number of activities (perhaps numbering in the hundreds) it may be prohibitively difficult to explicitly identify all of the paths through the network, but relatively easy to treat the completion of activities as events that may trigger the release of milestones. In this case an event-based simulation represents these paths implicitly, but is far easier to construct.

Let $\mathscr{I}(j)$ be the set of inbound activities to node j; for instance, $\mathscr{I}(c) = \{2, 3\}$ for the example in Fig. 3.2. Similarly, let $\mathscr{O}(j)$ be the outgoing activities from node j; thus, $\mathscr{O}(c) = \{5\}$. Finally, let $\mathscr{D}(\ell)$ be the destination node for activity ℓ, so that $\mathscr{D}(5) = d$. All of these sets would have to be known to construct the SAN, or could easily be extracted from a graphical representation. Now only a single type of event, which we call "milestone," is required to simulate the SAN. The milestone event has as arguments a target node j and an activity ℓ that is inbound to that node. It removes the completed activity from the set of inbound activities to that node, $\mathscr{I}(j)$, and when that set becomes empty it schedules all of the node's outbound activities $\mathscr{O}(j)$ to begin. In pseudocode, the event is as follows:

```
**event milestone** (activity $\ell$ inbound to node $j$)
$\mathscr{I}(j) = \mathscr{I}(j) - \ell$
if $\mathscr{I}(j) = \emptyset$ then
    for each activity $i \in \mathscr{O}(j)$
        schedule milestone(activity $i$ inbound to node $\mathscr{D}(i)$ to occur $X_i$ time later)
end if
```

This representation makes it much easier to program a stochastic activity network simulation, but actually increases the computer effort required to simulate it due to the need to manage an event calendar and execute the events.

Another feature of this example is that the goal is to estimate a probability $\theta = \Pr\{Y > t_p\}$. If t_p is large, such occurrences may be so rare that (say) $n = 1000$ replications might not generate even a single project that completes after t_p. Later in the book we will look for ways to make this simulation more statistically efficient, meaning that fewer replications will be needed to obtain the desired precision of the simulation estimate or higher precision will be obtained from the same number of replications.

3.5 Asian Option

A "call" option is a contract giving the holder the right to purchase a stock for a fixed "strike price" at some time in the future. If the stock's value increases well above the strike price, then the option is a good deal provided the purchase price for the option was not too high. If offered a call option with strike price K and maturity T on a stock that is currently trading at $X(0)$, how much should one be willing to pay for this option?

Valuing, pricing, and creating financial instruments that serve various needs are the purview of the field of financial engineering (FE). FE often requires stochastic simulation. The critical issue in FE relative to the other examples in this chapter is the need for highly precise estimates. A 5% error may be tolerable in estimating customer delay in queue, but entirely out of the question in pricing a financial product that may be sold in millions of units. The development in this section is based on Glasserman (2004, Chap. 1), an excellent text on simulation and FE.

The value of the asset on which the option is written (e.g., stock) is often modeled as a continuous-time stochastic process $\{X(t), 0 \le t \le T\}$. In the standard European option the payoff to the contract holder is

$$(X(T) - K)^+ = \max\{0, X(T) - K\}.$$

That is, if the value of the stock at maturity, $X(T)$, is greater than the strike price K then the owner can exercise the option—purchase the stock at price K—and immediately sell it for a profit of $X(T) - K$. Of course, if $X(T) \le K$, then the owner might as well just buy the stock on the open market and the option is worthless.

If the stochastic process $\{X(t), 0 \le t \le T\}$ is *geometric Brownian motion* (GBM, see Glasserman, 2004, Sect. 3.2), then the value of a European option can easily be computed using the celebrated Black–Scholes formula (e.g., Glasserman, 2004, Sect. 1.1). For the purpose of this book the important thing to know about GBM is that it is a continuous-time, continuous-state stochastic process; that is, $\{X(t), 0 \le t \le T\}$ has a value at all times t and $X(t) \in \Re$. This will cause some problems for simulation.

Since the European option can be valued without simulation, we take as our example a more difficult case:

Example 3.4 (Asian Option). Suppose we are offered an "Asian" option, which means that the payoff is $\left(\overline{X}(T) - K\right)^+$, where $\overline{X}(T)$ is an average of the stock's value over $0 \le t \le T$ (from now until maturity). Given the stock's current value

$X(0)$, a maturity T, a risk-free interest rate r (at which we could invest our money instead of buying the option), and a strike price K, the value of this option is

$$\nu = \mathrm{E}\left[\mathrm{e}^{-rT}\left(\overline{X}(T) - K\right)^{+}\right],$$

where the multiplier e^{-rT} discounts the value at time T back to time 0 (now). How can we use simulation to evaluate it?

This would appear to be the easiest example introduced in this chapter, since the following algorithm does it:

```
1. set $s = 0$
2. repeat $n$ times:

    a. generate $\overline{X}(T)$
    b. set $Y = \mathrm{e}^{-rT}\max\{0, \overline{X}(T) - K\}$
    c. set $s = s + Y$

3. estimate $\nu$ by $\widehat{\nu} = s/n$
```

The difficulty comes in Step 2a. We have not (yet) been precise about what is meant by $\overline{X}(T)$. Probably the most natural definition is

$$\overline{X}(T) = \frac{1}{T}\int_0^T X(t)\,\mathrm{d}t. \tag{3.13}$$

We have seen time averages before (e.g., Sect. 1.2), but this one is different: $X(t)$ does not just change at discrete points in time, it changes continuously. Thus, if we wanted to try to schedule events at which the state of $X(t)$ changes, there would be an uncountably infinite number of them, even on a finite time interval $[0,T]$. This issue arises in many FE simulations because continuous-time, continuous-state stochastic processes are often used to represent the values of underlying financial assets, such as stocks, over time.

A natural approximation for (3.13) is to take the interval $[0,T]$, divide it up into m steps of size $\Delta t = T/m$ and use

$$\widehat{\overline{X}(T)} = \frac{1}{m}\sum_{i=1}^{m} X(i\Delta t)$$

the average value of the stock at a set of monitoring times $\Delta t, 2\Delta t, \ldots, m\Delta t$.[5] This introduces discretization error, of course, but not much if Δt is not too large. On the other hand, if Δt is too small, then numerical round-off error can accumulate and the simulation can become quite slow. Thus, discrete approximations to continuous processes cannot be done without thought; see Sect. 5.3.

[5] Many Asian options are actually defined in this way with the time steps specified as part of the contract. For the discussion here we will assume that the time average $\overline{X}(T)$ is what is actually desired, however.

Suppose that we want to do a discrete approximation. If $X(t)$ is described by GBM with drift r (the risk-free interest rate) and volatility σ^2 (a measure of the variability of the asset's value), then it can be shown that for $0 = t_0 < t_1 < t_2 < \cdots < t_m = T$,

$$X(t_{i+1}) = X(t_i) \exp\left\{ \left(r - \frac{1}{2}\sigma^2 \right) (t_{i+1} - t_i) + \sigma \sqrt{t_{i+1} - t_i} Z_{i+1} \right\}, \qquad (3.14)$$

where $Z_1, Z_2, \ldots, Z_m$ are i.i.d. $\mathrm{N}(0,1)$. We will use this fact to simulate the Asian option in Chap. 4.

Exercises

1. Show that the limit (3.5) exists, and precisely what it is, if $Y_1, Y_2, \ldots$ are i.i.d. with finite variance ς^2.
2. For the $M/M/1$ queue, make a plot that shows the relative effort required (value of m) to obtain equal standard error $\sqrt{\mathrm{Var}\left(\overline{Y}(m)\right)}$ for $0.5 \le \rho \le 0.99$. Note that for the $M/M/1$ queue Eq. (3.4) simplifies because $\sigma = \tau$ for the exponential distribution.
3. Instead of trying to obtain the same standard error $\sqrt{\mathrm{Var}\left(\overline{Y}(m)\right)}$ for $0.5 \le \rho \le 0.99$, we could instead try to obtain the same *relative error*
$$\frac{\sqrt{\mathrm{Var}\left(\overline{Y}(m)\right)}}{\mathrm{E}(Y)}$$
for $0.5 \le \rho \le 0.99$. Make a plot similar to Exercise 2 for relative error.
4. Develop an event-based simulation of the $M/G/1$ queue similar to the TTF example. Keep track of the number of customers in the system.
5. Derive (3.8) via induction.
6. Show how to represent the simulation of the SAN in Fig. 3.2 using an event-based approach.
7. Sketch out the event logic you would use to simulate the $M(t)/M/\infty$ queue to keep track of both the average number of cars in the parking lot and the maximum number in the lot for a 24-h period. Do not be concerned with how to simulate the car arrival process, just assume that you have a method for generating the interarrival times between cars.
8. Using (3.14), add details to the algorithm for simulating an Asian option in Sect. 3.5 to produce a discrete approximation. Let m, the number of steps, be an input.
9. For the SAN example in Sect. 3.4, how many replications are needed to estimate θ with relative error 1% if $t_p = 5$?

Chapter 4
Simulation Programming with PythonSim

This chapter shows how simulations of some of the examples in Chap. 3 can be programmed in PythonSim. The goals of the chapter are to introduce PythonSim and to hint at the experiment design and analysis issues that will be covered in later chapters. This chapter can be skipped without loss of continuity. A complete listing of the PythonSim source code can be found at the book website.

4.1 A Python Primer

We present a brief primer on the basic Python programming language concepts necessary to use PythonSim, which is a simple Python module with objects and functions for object-oriented discrete-event simulation. This primer is a quick and informal introduction to coding with PythonSim; a thorough and comprehensive tutorial on Python is beyond the scope of this book. Our PythonSim implementation uses Python 3.

Python can be installed from source at http://www.python.org, and this installation also includes the commonly used Python package manager called `pip`. The ubiquitously used `numpy` and `pandas` packages, which we utilize in PythonSim, can be installed with `pip`. There are many package managers and integrated development environments (IDEs) or editors for Python, as well as multiple ways to run Python code, but these are not discussed in this primer. We recommend more complete tutorials, such as the guides on http://www.python.org for those unfamiliar with programming languages or Python.

We use Python due to its simplicity and popularity. Python is a high-level language and an interpreted rather than compiled language. Python is widely regarded as easy to learn due to its simple syntax, and Python programs are generally considered shorter and more readable than C, C++, or Java counterparts because Python

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_4

does not require variable or argument declarations, and it uses indentation rather than brackets. This simplicity is ideal for our goal of teaching the fundamentals of programming a simulation, rather than teaching professional coding practices. Additionally, Python has many popular libraries, some of which support integration with other languages such as C or C++. Python is widely used in applications such as machine learning, data science, finance, and scientific computing, and many of the Python principles we discuss in the context of PythonSim are useful in these applications as well.

4.1.1 Variables and Data Types

Python variable names are case-sensitive, can only use alphanumeric characters and underscores, and cannot start with a number. Variables are assigned values using the assignment operator `=`, e.g., `x = 5` creates a variable `x` with the value 5. This x is an integer (type `int`) variable. Assigning `x = 5.0` rather than `x = 5` results in x being a float (type `float`). When working with numbers, the arithmetic operators `+, -, *, /, %,` and `**` for addition, subtraction, multiplication, division, modulus, and exponentiation, respectively, behave in the usual way. Specifically, in Python 3, `/` is for float division (returns a float) and `//` is for integer division or floor division (returns the closest integer less than or equal to the expression). We also sometimes use `n += 1` as a compact way of incrementing a variable, and this expression performs the same action as `n = n + 1`.

In addition to integers and floats, Python uses other data types such as strings (type `str`), Booleans (type `bool`), and lists (type `list`). This is not an exhaustive list of data types. Notice that if `x` is a defined variable, then `type(x)` returns the Python type of `x`. Python strings are enclosed in single or double quotations. Each character in a string is from the unicode character set, and strings can be empty (e.g., `""`) and can also contain spaces (e.g., `"Hello there?"`).

Strings are concatenated (joined together) with the `+` operator and can be repeated using the `*` operator, as we show in the code snippet below. Notice that attempting to "add" variables or values of different types, such as `2 + "banana"`, does not work. The code snippet also demonstrates the `print()` function in Python, which takes any object and converts it into a string before writing it to the screen. Notice that we can also pass an *expression* (e.g., `s1 + s2`) as an argument to `print()` and Python evaluates the expression for us, as is the case for Python functions in general. The result of `s1 + s2` is `"Hello, world!"` and `s3` holds the string `"abcdabcd"`, i.e., `"abcd"` repeated twice.

```
s1 = "Hello, "

s2 = ""
s2 += "world!"

s3 = "abcd" * 2
```

```
print(s1 + s2)
print(s3)
```

The built-in Python data type `lists` stores multiple items, which can be of different types, and separates the items with commas. `[]` is an empty list. An element of a list can also be a list. If `ls` is a list, then `ls.append(x)` appends the object `x` to the end of the list `ls`, modifying `ls` in place. For lists, the + and `*` operators can be used to join and multiply lists, working just as they do for strings. The following three code snippets result in the same value of the `ls` variable.

```
# Code Snippet 1
ls = []
ls.append(1)
ls.append("two")

# Code Snippet 2
ls = [1, "two"]

# Code Snippet 3
ls = [1] + ["two"]
```

Notice that `ls` is a list of both integer and string types. The code above also shows comments in Python. Lines starting with a # are ignored by the interpreter and do not affect the program. If `ls` is a list, then `ls.pop()` both removes and returns the last element of the list. The code below results in `ls` having value `[1,2]` and `x` having value `3`.

```
ls = [1,2,3]
x = ls.pop()
```

The built-in function `len()` can act on both strings and lists and returns the length of the argument, i.e., the number of characters in a string or the number of items in a list. Empty strings and empty lists have length `0`.

Next, we discuss indexing and slicing. We focus on lists, but the indexing and slicing syntax and logic work analogously for strings and some other Python data types as well. In Python, we start indexing the number of items in a list at `0`, so that the first item has index `0`. If `ls` is a list and `i` is a nonnegative integer, then `ls[i]` returns the item in the list at index `i`, provided that `i` is *strictly* less than the length of the list `ls`. Notice that `ls[len(s)]` throws an out of range error, since `ls[len(s)-1]` is actually the last character, due to indexing starting at `0`. Python supports negative indexing, which indexes a list *from the end*, starting with index `-1` as the last element. Python also supports slicing, so that if `ls` is a list and `i` and `j` are integers, then `ls[i:j]` returns the "slice" (subset) of `ls` starting from index `i` and ending just before index `j`.

However, strings are immutable while lists are mutable. Assigning a new character to an indexed position in a string creates an error. But the same action is valid for lists. For example, the following code does not work and creates an item assignment error.

```
s = "bat"
s[0] = "c"
```

Meanwhile, the following code works.

```
ls = ["apples", "oranges"]
ls[1] = "watermelon"
```

4.1.2 Indentation, Conditions, and Loops

Here, we discuss conditions and loops in Python, as well as the Python indentation required for writing them.

Python separates blocks of code using indentation. Python programs run properly if the number of spaces is at least 1 and remains consistent throughout the program, but the convention is to use 4 spaces as an indent in Python. Suppose that `x` is an integer variable, and consider the following code snippet, which demonstrates the use of indentation in a series of conditional statements.

```
x = 10

if x < 10:
    print("x is less than 10.")
elif x == 10:
    print("x is equal to 10.")
else:
    print("x is greater than 10.")
```

In Python, `if`, `elif`, and `else` are used for decision-making and cause a program to execute code only if a certain condition is true. For completeness, we note that Python interprets expressions that evaluate to a nonzero value as `True` and expressions that evaluate to `None` or `0` as `False`. In the code above, the print function in the body of the `if` statement is only executed if `x < 10` is `True`. If `x < 10` is `False`, then the program moves to the following `elif` statement and executes its body if `x == 10` evaluates to `True`. Statements starting with `elif` (short for "else if") are optional but must come after an `if` statement or another `elif` statement. Multiple `elif` statements can be used, but if multiple `elif` statements are `True`, only the body of the first `True` `elif` statement is executed. If `x == 10` is `False`, then the program moves to the `else` statement and executes its body. An `else` statement is optional, but it must come after an `if` or `elif` statement, which also means there cannot be more than one `else` statement in a row. We can also nest `if`, `elif`, and `else` conditions. Inner conditions must have additional indentation and obey the same logical rules as the outer conditions.

In the code above, `x < 10` and `x == 10` are examples of logical conditions in Python. Python supports the usual logical operators `==`, `!=`, `<`, `<=`, `>`, and `>=`. Notice that *two* equal signs `==` are used for checking equality in a condition, since the use of only one equality sign `=` is the assignment operator. Conditions such as `x < 10` and `x == 10` evaluate to `True` or `False`, which are keywords of the Boolean data type in Python (type `bool`). Conditions can be combined with the `and` and `or` keywords acting as logical operators. In the code below, `x` is only printed if *both* conditions `x %10 == 0` and `x < 100` are `True`, and a nested `if` statement prints another message if in addition to the two previous conditions holding, `x == 50` holds as well.

```
if x %10 == 0 and x < 100:
    print(x)
    if x == 50:
        print("x equals 50.")
```

Logical conditions are important for `while` loops, which repeat the execution of code, while a certain condition is true. Loops are crucial, particularly in simulation programming, because they allow us to execute statements multiple times. The following simple `while` loop sequentially increments and prints `n` until `n <= 10` is no longer `True`. Notice that as with conditional statements, the body in a loop must be indented. We must be careful with `while` loops, because if the `while` condition is always `True`, then the program will loop infinitely and will never terminate.

```
n = 0
while n <= 10:
    n += 1
    print(n)
```

Python `for` loops behave similarly as `while` loops but repeat the execution of code a certain number of times. Python `for` loops are commonly used with the built-in `range` function. This combination is useful for iterating over multiple replications in a simulation model. If `n` is a positive integer, then `range(n)` returns a sequence of integers from `0` up to but not including `n`. The following code successively prints `0`, `1`, ...,`9`.

```
for i in range(10):
    print(i)
```

Python supports nesting `while` and `for` loops within each other, and each nested loop requires more indentation. Conditional statements can also be nested within loops, and vice versa. Often, conditional statements within a loop are used to trigger the control statements `break` and `continue`, which modify normal loop execution. The statement `break` immediately terminates the loop, and `continue` skips any subsequent code in the body of the loop and sends the program to test the loop condition. As an exercise, work through the logic of the code snippet below, which successively prints 1, 2, 4, and 5.

```
n = 0
while n <= 10:
    n += 1
    if n == 3:
        continue
    print(n)
    if n == 5:
        break
```

Finally, in Python, one way to wrap long lines is by creating line breaks within parentheses, brackets, or braces. In the code below, the first and second code snippets, as labeled using Python comments, behave identically. In the case of the second code snippet, by using Python's implied line continuation within the parentheses, we can break a long line into two more manageable lines.

```
# Code Snippet 1
if SomeConditionA == True and SomeConditionB == True:
    print("Both conditions hold.")

# Code Snippet 2
if (SomeConditionA == True
    and SomeConditionB == True):
        print("Both conditions hold.")
```

4.1.3 Functions, Scope, and Modules

Python functions are created using the def keyword, followed by the name of the function, and then comma-separated parameter names enclosed in parentheses, followed by a colon. The body of the function must be indented. Functions can send objects back to the caller using the return keyword, although functions do not require return statements.

Consider the code below, in which we define a function called CutInHalf with parameter m. Triple quotation marks support multiline comments and also demarcate a docstring at the beginning of a function, which describes a function's behavior. Function declarations have *parameters*, and functions are passed specific *arguments* when called. In the last line in the code below, we call CutInHalf(10) with argument 10 and assign the return value to a new variable n.

```
def CutInHalf(m):
    '''Divides a number by 2'''
    return m/2

n = CutInHalf(10)
```

The scope of a variable concerns its visibility and accessibility. Python variables created outside of a function have global scope and are visible from the code anywhere in a program. When a function is called (not when it is defined), any variables defined inside the function have local scope and are only visible from the code inside the function. After the function call, the corresponding local scope is destroyed.

Python looks up variable names by first checking local scopes, then enclosing scopes (which we do not elaborate on here but involves nested functions), and then the global scope. Even though global names can be accessed anywhere, they cannot be modified within a function. To illustrate this point, we provide a code example below, and this code also demonstrates the declaration of a function without parameters and without a return statement. Although `x` is the name of a global variable, calling the function `ChangeTo10()` cannot modify that global variable. Instead, what `ChangeTo10()` does is create a new variable called `x` in the function's local scope. The global variable with the same name remains unchanged, which is why the final print statement, despite being after a `ChangeTo10()` function call, prints `0` rather than `10`.

```
x = 0

def ChangeTo10():
    x = 10
    print(x)

# This prints 10
ChangeTo10()

# Global variable x is unchanged
# This prints 0
print(x)
```

The following modification of `ChangeTo10()` does successfully update the global variable `x` to its new desired value `10`, but for more complicated code, returning and assigning values in this way can result in overly lengthy and cumbersome code.

```
def ChangeTo10():
    x = 10
    return x

x = ChangeTo10()
```

The `global` keyword tells Python to look for `x` in the global scope, rather than create a new local variable, and the following modification of `ChangeTo10()`

also successfully updates the global variable x. However, this practice is generally discouraged because it makes the code more complicated and harder to maintain.

```
def ChangeTo10():
     global x
     x = 10
```

A better practice for handling global variables that need to be modified within function calls is to define such variables in a separate file (module), which is then imported. This practice is used for a simulation model's clock time and is detailed in Sect. 4.1.5.

4.1.4 Modules, Packages, and Writing Output

A module is a Python file containing definitions and statements. A package is a collection of modules, usually following some sort of organization or hierarchy. Both modules and packages can be loaded into a Python program using the `import` keyword, and modules and packages are useful for structuring and reusing code.

PythonSim uses the built-in package `math` and the two popular packages `numpy` and `pandas`, which can be downloaded using `pip` after installing Python from source at http://www.python.org.

Dot notation is Python's way of accessing objects in another module or package. For example, after we call `import math`, we can use `math.inf` to represent infinity in logical operations and conditions. When called on a number in a suitable domain, the functions `math.sin`, `math.exp`, `math.log`, and `math.sqrt` give access to the sine function, exponential function, logarithmic function, and square root function, respectively.

The `numpy` package is a very powerful package supporting `numpy` arrays (type `numpy.ndarray`), which have similar functionality as lists but are optimized for scientific computing performance. A proper discussion of `numpy` is beyond the scope of this primer, but we use `numpy` in PythonSim programs to apply mathematical operations element-wise to a list of numbers; an example is shown below.

```
data = [1,2,3,4,5]

# This statement works
squareroot_data = numpy.sqrt(data)

# This statement does not work
squareroot_data = math.sqrt(data)
```

The `pandas` package is a library for data analysis on numerical tables. We use `pandas` for its dataframes (type `pandas.core.frame.DataFrame`), which have useful methods for statistics and writing data to comma-separated values (.csv) files. Writing to .csv files is useful for saving data for future analysis, and .csv files can also be imported as tables into Excel.

Suppose that `Statistic1` and `Statistic2` are lists of equal length. For example, in a simulation setting, each item of `Statistic1` and `Statistic2` corresponds to a statistic from an i.i.d. simulation replication. The following code demonstrates how to write simulation data to a .csv file using `pandas`. The strings `"Statistic1"` and `"Statistic2"` become the column names in a dataframe, and `Statistic1` and `Statistic2` become the data columns. Here, the input to creating a dataframe is a dictionary (type `dict`), a data structure that stores data in pairs of the form key:value. Elaborating in detail about dictionaries is outside the purpose of this PythonSim primer, but we note that dictionaries are powerful tools supporting fast lookup of values based on keys. Keys index a dictionary and must be unique (no duplicates allowed) and of an immutable data type (such as a string). This structure is natural for building tables of data which are organized by columns each corresponding to a unique statistic. The last line in the code below writes the dataframe to a .csv file called `"output.csv"` in the current working directory, with elements separated by commas.

```
output = pandas.DataFrame(
    {"Statistic1": Statistic1,
    "Statistic2": Statistic2})
output.to_csv("output.csv", sep=",")
```

The built-in Python function `len()` can take in a dataframe to return its number of rows. If `output` is a dataframe, then `output.mean()` and `output.var()` return the mean and sample variance of each column's values.

4.1.5 Object-Oriented Programming and Classes in PythonSim

PythonSim consists of classes, functions, and random-number generation tools, all of which are entirely open source and can be modified to suit the user. PythonSim provides tools to aid in developing discrete-event simulations and is designed to be easy to understand and use, but not necessarily efficient.

In this section, we discuss object-oriented programming (OOP) in Python and introduce the classes in PythonSim. We cannot cover all of the important topics in object-oriented programming, but we briefly describe the OOP concepts that PythonSim exploits. OOP is incredibly valuable for simulation programming, because discrete-event simulations often contain multiple instances from the same template, so it makes sense to structure our programs around classes, which are blueprints for creating similar objects. For example, discrete-event simulations might track multiple continuous-time statistics, such as a number of customers and a number of busy servers in the system. Although the number of customers and the number of busy servers are distinct statistics that track different quantities and take different values in a simulation, the machinery for recording and computing these statistics is the same. Therefore, we can bundle this machinery in a class for continuous-time statistics and let the number of customers and the number of busy servers be instances of this class. Furthermore, OOP leads to organized and readable code, which supports large-scale simulation models. Classes allow attributes to be grouped and created in one place, and we can add new functionality to classes without breaking previous code.

The classes in PythonSim reside in the file `SimClasses.py`. This file acts like a module and can be imported using `import SimClasses` at the beginning of a simulation program. To access the classes in `SimClasses`, we use Python's dot notation. For example, `SimClasses.Clock` accesses the `Clock` variable in the `SimClasses` module. In each simulation, we need the simulation clock to be globally viewable and modifiable. Because Python only creates one instance of each imported module, `SimClasses.Clock` can be accessed and changed from any file that uses `import SimClasses`.

Below, we list the classes in PythonSim that are the building blocks for a PythonSim discrete-event simulation.

- `CTStat` objects record continuous-time statistics.
- `DTStat` objects are companions to `CTStat` objects and behave similarly but record discrete-time statistics.
- `Entity` objects model transient items, such as transactions or customers that pass through the system.
- `FIFOQueue` objects hold `Entity` objects in first-in-first-out order.
- `Resource` objects represent scarce quantities such as workers, machines, and computers that are needed to serve or process an `Entity` in some way.
- `EventNotice` objects model the state-changing events that drive a discrete-event simulation.
- `EventCalendar` objects are lists of event notices ordered by time.

The `SimClasses` module also includes `Activity` and `Node` classes, but those are specific to simulating a stochastic activity network (SAN) and are discussed in Sect. 4.4.2.

The code for the class `CTStat` is shown in Fig. 4.1, and we use the `CTStat` to illustrate some key concepts in Python OOP. We create a class in Python using the `class` keyword, followed by the class name and a colon. Notice that the body of the class is indented.

`InstanceList` is created outside of the method `__init__()` and is a class attribute, which means it is shared across all instances of the class. We can access it using `SimClasses.CTStat.InstanceList`, and we can also assign it a new value. In the `__init__()` method, we define instance attributes, which can have different values across different instances of the class. The `__init__()` method can be defined with any number of parameters, but the first parameter must always be `self`. The `__init__()` method initializes each new instance, because it is run each time a new instance is created and sets the default values for the instance's attributes. The `self` keyword allows an instance to refer to itself. We also define the methods `Record()`, `Mean()`, and `Clear()`, which are instance methods that take `self`

```
class CTStat:

    InstanceList = []

    def __init__(self):

        self.Area = 0.0
        self.Tlast = 0.0
        self.TClear = 0.0
        self.Xlast = 0.0
        self.Max = -math.inf
        self.Min = math.inf

        self.__class__.InstanceList.append(self)

    def Record(self,X):

        self.Area += self.Xlast * (Clock - self.Tlast)
        self.Tlast = Clock
        self.Xlast = X

        if X > self.Max:
            self.Max = X

        if X < self.Min:
            self.Min = X

    def Mean(self):

        mean = 0.0
        if (Clock - self.TClear) > 0.0:
           mean = ((self.Area + self.Xlast * (Clock - self.Tlast))
            / (Clock - self.TClear))
        return mean

    def Clear(self):

        self.Area = 0.0
        self.Tlast = Clock
        self.TClear = Clock
```

Fig. 4.1 PythonSim CTStat class for continuous-time statistics

as the first parameter but can also include any number of additional parameters. The last line in the `__init__()` method body adds the newly created instance to the class attribute `InstanceList`. This allows us to keep track of all continuous-time statistics in a simulation program, so we can clear them between replications.

The following code imports `SimClasses` and instantiates a new instance of the class `CTStat` using the call `SimClasses.CTStat()`.

```
import SimClasses

NumCustomers = SimClasses.CTStat()
```

If __init__() were to take parameters other than self, we would call SimClasses.CTStat() with additional arguments to pass to these parameters. Now, NumCustomers is a CTStat object. Whenever we create a new instance of a class, this instance is its own unique copy with specific characteristics that we can modify. Using dot notation, we can access an instance's attributes and methods. For example, NumCustomers.Max gives us the instance's attribute Max, and we can also assign NewCustomers.Max a new value. If N is an integer variable, then NumCustomers.Record(N) calls the instance method Record() with the argument N and records N as the current state (the current number of customers). When instantiating an instance or calling an instance method, there is no need to pass self as parameter because this is handled automatically.

In general, whenever the value of the variable of interest changes, the Record() method of the relevant CTStat object is employed to record the change (which means the method is called just after the change occurs). The Mean() method returns the continuous-time average up through the current time in the simulation but does not update any variable values. DTStat objects behave very similarly, and after each new observation of interest (e.g., a waiting time or a service time has just finished), we call the Record() method of the relevant DTStat to record this new observation. The Mean() method returns the average of the discrete-time observations accumulated up through the current time.

After introducing some important OOP Python syntax, we conclude this section by pointing out some important details about the PythonSim classes. Entity instances are typically dynamically generated through a simulation run. For example, we might generate a new Entity instance representing a customer every time an arrival event occurs. Entity instances have attributes that they carry with them; by default each instance has an attribute called CreateTime, which is set to the value of Clock at its creation time. The FIFOQueue and Resource classes each have an InstanceList class attribute, just like the CTStat and DTStat classes, so that each queue and resource instance can be properly reset between simulation replications. FIFOQueue objects have a built-in CTStat for work-in-process (WIP), i.e., the number in the queue, and Resource objects have a built-in CTStat for the number of busy resource units. Each FIFOQueue and Resource instance's built-in CTStat is automatically added to SimClasses.CTStat.InstanceList as well. EventNotice instances, like Entity instances, are typically dynamically generated as the simulation progresses and are managed by an EventCalendar instance. Typically, we have one EventCalendar instance for our simulation, and its attribute ThisCalendar starts off as an empty list. EventNotice objects are added to the EventCalendar instance and removed in chronological order when the simulation clock time reaches their occurrence times.

In the following section, we discuss the PythonSim functions that act on PythonSim class instances as well as event-scheduling logic.

4.1.6 *Functions, Random-Number Generation, and Event Scheduling in PythonSim*

The modules `SimFunctions.py` and `SimRNG.py` contain Python functions for managing events and initializing simulation replications, as well as random-number generation. Using `import SimFunctions` and `import SimRNG` at the beginning of a program allows access to both modules. The random-number and random-variate generation routines are Python translations of the corresponding routines in simlib (Law, 2007), which is written in C.

Here is a brief overview of the functions in PythonSim:

`SimFunctions.SimFunctionsInit()` initializes PythonSim for use by "reseting" key objects, typically called before the start of each replication. Resets the simulation clock, empties the event calendar, empties all queues, reinitializes resources, and clears continuous-time and discrete-time statistics.
`SimFunctions.Schedule()` schedules future events on an event calendar.
`SimFunctions.SchedulePlus()` has the same functionality as `Schedule`, with the additional capability of storing an object along with the scheduled event.
`SimFunctions.ClearStats()` clears continuous-time and discrete-time statistics.
`SimRNG.InitializeRNSeed()` initializes the random-number generator, typically called only once in a simulation.
`SimRNG.Expon()` generates exponentially distributed random variates.
`SimRNG.Uniform()` generates uniformly distributed random variates.
`SimRNG.RandomInteger()` generates a random integer.
`SimRNG.Erlang()` generates Erlang distributed random variates.
`SimRNG.Normal()` generates normally distributed random variates.
`SimRNG.Lognormal()` generates lognormally distributed random variates.
`SimRNG.Triangular()` generates triangularly distributed random variates.

The random-variate generation functions take two types of arguments: parameters and a random-number stream; the random-number stream is always the last argument. For instance,

```
X = SimRNG.Uniform(10, 45, 2)
```

generates random variates that are uniformly distributed between 10 and 45, using stream 2.

As you already know, the pseudorandom numbers we use to generate random variates in simulations are essentially a long list. Random-number streams are just different starting places in this list, spaced far apart. The generator in PythonSim (which is a translation of the generator in Law (2007)) has 100 streams. Calling `SimRNG.InitializeRNSeed` sets the initial position of each stream. Then, each subsequent call to a variate generation routine using stream # advances stream # to the next pseudorandom number. The need for streams is discussed in Chap. 7,

```
for reps in range(NumReps):

    SimFunctions.SimFunctionsInit(Calendar)
    <schedule events>

    while <some stopping condition not yet met>:

        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime

        if NextEvent.Type == "A":
            <do function A>

        elif NextEvent.Type == "B":
            <do function B>

        <conditional statements for other event types>

    <record replication statistics>
```

Fig. 4.2 Pseudocode outline for simulation experiment main loop

but it is important to know that any stream is as good as any other stream in a well-tested generator.

Next, we discuss a general recipe for building discrete-event simulations in PythonSim, putting the tools from the `SimClasses`, `SimFunctions`, and `SimRNG` modules all together. The code for the examples in Sects. 4.2–4.6 is loosely divided into different parts:

- Imports: importing the PythonSim modules `SimClasses`, `SimFunctions`, and `SimRNG`, and importing packages such as `math`, `pandas`, and `numpy`.
- Initialization and parameters: initializing a random-number generator, creating an event calendar, creating discrete-time and continuous-time statistics as well as lists to hold these statistics after each replication, and setting values such as parameters in distributions and a number of simulation replications to run.
- Simulation functions: defining "helper functions" triggered when certain event types occur.
- Simulation experiment: the main loop for running multiple replications of a simulation experiment.
- Simulation output: writing data to .csv files and printing relevant statistics.

In Fig. 4.2, we provide pseudocode outlining the general structure of a simulation experiment main loop. Suppose that `NumReps` is a positive integer for the desired number of simulation replications and `Calendar` is an instance of `SimClasses.EventCalendar`. The angle brackets are informal pseudocode, intended to summarize code that varies based on the specific simulation program.

```
def Schedule(calendar,EventType, TimeUntilEvent):
    addedEvent = SimClasses.EventNotice()
    addedEvent.EventType = EventType
    addedEvent.EventTime = SimClasses.Clock + TimeUntilEvent
    calendar.Schedule(addedEvent)
```

Fig. 4.3 `SimFunctions.Schedule()` function

The simulation main loop in the examples of Sects. 4.2–4.6 has an outer loop for multiple simulation replications and an inner loop cycling through events within each simulation replication. At the beginning of each simulation replication, calling `SimFunctions.SimFunctionsInit()` and passing our event calendar instance `Calendar` "resets" our simulation so that it starts fresh for the new replication. Then, we schedule events, such as the first event in the simulation (e.g., the first customer arrival) and events with occurrence times that we know up front (e.g., a staff shift change time, a warm-up period, a simulation stopping time).

While a stopping condition is not yet met, the program cycles through events on the event calendar. `NextEvent = Calendar.Remove()` removes and returns the next event from `Calendar`. The statement `SimClasses.Clock = NextEvent.EventTime` advances simulation clock time to this event's occurrence time. Then, the program executes certain actions depending on the type of the event. For example, an "arrival" event might trigger creating another `Entity` instance and scheduling another arrival event on `Calendar`. Commonly, these actions are packaged in a previously defined helper function for more organized and readable code. After the simulation replication, we record replication statistics.

We conclude this section by discussing `SimFunctions.Schedule()`, which is shown in Fig. 4.3. This function is fundamental to a discrete-event simulation because it creates and schedules the events that drive our simulation. In PythonSim, we schedule an event by calling `SimFunctions.Schedule()` and passing our event calendar object, a string for the event type, and a time until event. Notice that PythonSim requires the user to decide on the base time unit in the simulation and to use it consistently throughout. `SimFunctions.Schedule()` creates a new `EventNotice` instance, which is inserted into the event calendar in chronological order.

4.2 Simulating the $M(t)/M/\infty$ Queue

Here, we consider the parking lot example of Sect. 3.1, a queueing system with a time-varying car arrival rate, exponentially distributed parking time, and an infinite number of parking spaces. The event-driven simulation program consists of initialization and parameter specifications (Fig. 4.4), a function to generate car arrivals (Fig. 4.5), event routines (Fig. 4.6), a main program loop (Fig. 4.7), and some output printing and writing (Fig. 4.8). There are two important aspects of PythonSim that are illustrated by this example: event scheduling and keeping track of quantities such as time-averaged statistics.

We begin by initializing the parking lot simulation and specifying parameters, as shown in Fig. 4.4. The key state variable to track in the simulation is the number of cars in the lot. We initialize a continuous-time statistic `NumCars` to track the time-averaged number of cars in the lot. We can also use `NumCars` to easily collect auxiliary statistics, such as the maximum value observed so far and the number of cars in the lot at the end of the simulation run after 24 h. We initialize lists to hold the results of each desired statistic after each independent simulation replication. We also specify a `MeanParkingTime` of 2 and an `EndSimulationTime` of 24, where the units are in hours. We can choose our own units, but we must make sure all functions and quantities reference consistent units.

There are two events that affect the number of cars in the parking lot: the arrival of a car and the departure of a car. We define functions `Arrival` and `Departure`, which are called whenever arrival and departure events occur, respectively. These functions are defined in Fig. 4.6. Since only one car arrives and departs at each arrival and departure event, respectively, we can easily keep track of the total number of cars in the parking lot. Upon an arrival, we record `NumCars.Record(NumCars.Xlast + 1)`, since one car just arrived, and we make an analogous record for each departure event.

The function `NSPP`, which stands for nonstationary Poisson process, is recruited during an arrival event to compute interarrival times according to our desired nonstationary arrival rate. Code is shown in Fig. 4.5. The formal definition of a nonstationary Poisson process is a topic of Chap. 6. However, here we provide an intuitive justification for how `NSPP` works.

Recall that the arrival rate (in cars per hour) to the parking lot in Example 3.1 is modeled by the function $\lambda(t) = 1000 + 100\sin(\pi t/12)$. To make our simulation execute more quickly for the purpose of this introduction, we scale down that rate by a factor of 10 and instead use $\lambda(t) = 100 + 10\sin(\pi t/12)$. This new arrival rate varies between 90 and 110 cars per hour, depending on the hour of the day t.

A stationary Poisson process has times between arrivals that are exponentially distributed with a fixed rate λ (or equivalently, a constant mean time between arrivals $1/\lambda$). The inverse cdf method for generating exponentially distributed random variates is described in Sect. 2.2. The maximum arrival rate for $\lambda(t)$ is 110 cars per hour, so `NSPP` generates possible arrivals using a stationary arrival process with rate $\lambda = 110$. To achieve the time-varying arrival rate, it only accepts a *possible* arrival at time t as an *actual* arrival with probability $\lambda(t)/\lambda$. When $\lambda(t) = 110$, a possible arrival is guaranteed to be an actual arrival, and when $\lambda(t) = 90$, a possible arrival only has a $90/110$ chance of becoming an actual arrival. Chapter 6 provides a rigorous explanation of how this "thinning" method generates a nonstationary Poisson process with the desired rate.

While there is only one upcoming arrival event at any point in time, handling departure events is trickier. There are not a fixed number of departure events because there are as many pending departure events as there are cars in the lot. Therefore, having a unique variable to represent the scheduled time of each pending event, as

```
# PythonSim imports
import SimClasses
import SimFunctions
import SimRNG

# Python package imports
import math
import pandas

# Initialization
SimClasses.Clock = 0
ZSimRNG = SimRNG.InitializeRNSeed()
Calendar = SimClasses.EventCalendar()

# Create CTStat object for number of cars in lot
NumCars = SimClasses.CTStat()

# Example-specific statistics
# NumCarsAvg: list of average queue length in each replication
# NumCarsMax: list of max queue length in each replication
# NumCarsAt24Hrs: list of queue length at simulation end
#   in each replication
NumCarsAvg = []
NumCarsMax = []
NumCarsAt24Hrs = []

# Parameters to specify
MeanParkingTime = 2.0
EndSimulationTime = 24

# Number of simulation experiment replications
NumReps = 10
```

Fig. 4.4 Initializing the parking lot simulation

was used for the TTF example in Chap. 2, does not work here. Discrete-event simulations can easily accommodate the unlimited number of pending departure events, by dynamically scheduling a departure event at each arrival event. Specifically, at each arrival event, we schedule a departure event corresponding to the time that this newly arriving car departs, as shown in the `Arrival` function in Fig. 4.6.

The main loop for simulation of the $M/M/\infty$ queue is displayed in Fig. 4.7. In addition to arrival and departure events, we also have a third type of event to stop the simulation after 24 h. Figure 4.8 provides code for generating a dataframe from our lists of statistics `NumCarsAvg`, `NumCarsMax`, and `NumCarsAt24Hrs`.

Figure 4.9 shows a histogram of the 1000 daily averages of the number of cars in the parking lot obtained by running the simulation for 1000 replications; the overall average of these averages is 184.2 ± 0.3 cars, where the "± 0.3" part comes from a 95% confidence interval on the mean (confidence intervals are a subject of Chap. 7). Thus, the simulation provides a pretty precise estimate of the time-average mean

```
def NSPP(Stream):
    '''
    Generates next interarrival time until next nonstationary
        Poisson arrival using thinning method

    Input:
        Stream: integer, corresponding to random number stream

    Output:
        nspp: float, positive, time until next nonstationary
            Poisson arrival
    '''

    PossibleArrival = SimClasses.Clock
    Ratio = 0
    while SimRNG.Uniform(0, 1, Stream) >= Ratio:
        PossibleArrival += SimRNG.Expon(1.0/110, Stream)
        Ratio = (100 + 10 * math.sin(math.pi
            * PossibleArrival / 12)) / 110.0

    nspp = PossibleArrival - SimClasses.Clock
    return nspp
```

Fig. 4.5 Function to generate interarrival times to the parking lot

```
def Arrival():
    '''
    Called when an arrival event occurs
    Increments NumCars count by 1 and records
    Schedules next arrival event
    Also schedules departure event of car that just arrived
    '''

    NumCars.Record(NumCars.Xlast + 1)

    SimFunctions.Schedule(Calendar,"Arrival", NSPP(1))

    SimFunctions.Schedule(Calendar,
        "Departure",SimRNG.Expon(MeanParkingTime, 2))

def Departure():
    '''
    Called when a departure event occurs
    Decrements NumCars count by 1 and records
    '''

    NumCars.Record(NumCars.Xlast - 1)
```

Fig. 4.6 Event routines for the parking lot simulation

```
for Reps in range(0,NumReps):

    # Initialize simulation objects and schedule first event
    #   and end-of-simulation event
    SimFunctions.SimFunctionsInit(Calendar)
    SimFunctions.Schedule(Calendar,"Arrival",NSPP(1))
    SimFunctions.Schedule(Calendar,
     "EndSimulation", EndSimulationTime)

    # Main simulation loop
    while Calendar.N() > 0:
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
        if NextEvent.EventType == "Arrival":
            Arrival()
        elif NextEvent.EventType == "Departure":
            Departure()
        elif NextEvent.EventType == "EndSimulation":
            break

    # Add replication statistics to respective lists
    NumCarsAvg.append(NumCars.Mean())
    NumCarsMax.append(NumCars.Max)
    NumCarsAt24Hrs.append(NumCars.Xlast)
```

Fig. 4.7 Main program for the parking lot simulation

```
# Create dataframe for replication statistics
#   and write to csv file
output = pandas.DataFrame(
    {"NumCarsAvg" : NumCarsAvg,
    "NumCarsMax": NumCarsMax,
    "NumCarsAt24Hrs" : NumCarsAt24Hrs})
output.to_csv("MMInfinity_output.csv", sep=",")

print("Replication Stats")
print(output)

print("Means")
print(output.mean())
```

Fig. 4.8 Printing and writing output

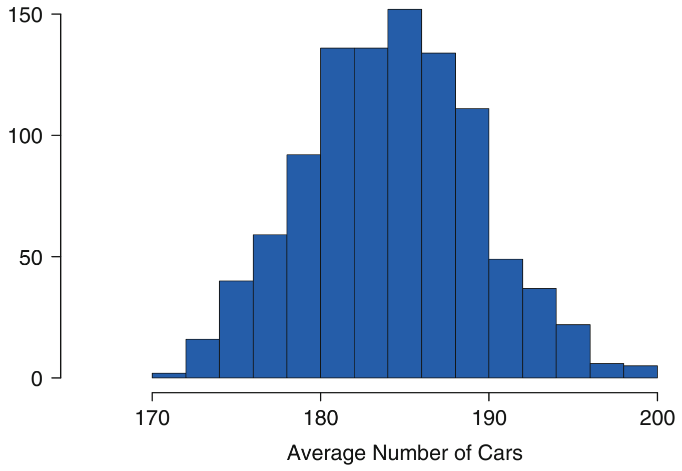

Fig. 4.9 Histogram of the daily average number of cars

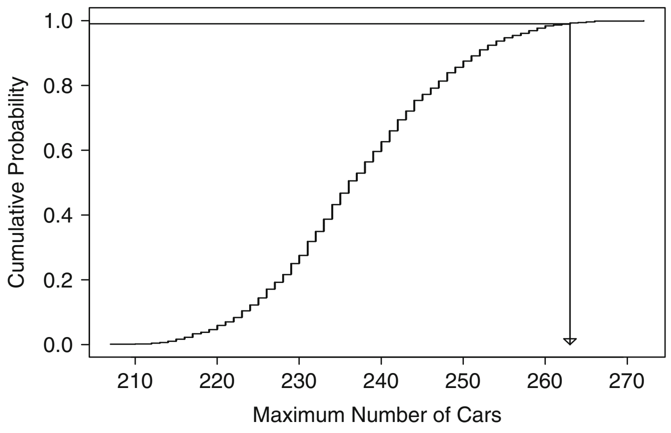

Fig. 4.10 Empirical cdf of the daily maximum number of cars in the parking lot

number of cars that would be found in the (conceptually) infinite-size garage during a day.

The histogram shows that the average number of cars in the garage can vary substantially from day to day, so we certainly would not want to build a garage with a capacity of, say, 185 cars. Furthermore, averaging over the day masks the largest number of cars in the garage during the day and that number is more useful for selecting a finite capacity for the garage.

Suppose that we want the parking lot to be of adequate size 99% of the time. Since we record the maximum size on 1000 replications, we could use the 990th value (sorted from smallest to largest) as the size of the garage, which turned out to be 263 cars in this simulation. Figure 4.10 shows the empirical cumulative distribution (ecdf) of the 1000 maximums recorded by the simulation. The ecdf treats each observed value as equally likely (and therefore as having probability $1/1000$) and plots the sorted maximum values on the horizontal axis and the cumulative probability of each observation on the vertical axis. The plot shows how the cumulative probability of 0.99 maps to the value of 263 cars, which might be a very reasonable capacity for the garage. Putting a confidence interval on this value is quite different from putting one on the mean and will be discussed in Chap. 7. Without a confidence interval (or some measure of error), we cannot be sure if 1000 replications are really enough to estimate the 99th percentile of the maximum.

4.2.1 Issues and Extensions

1. The $M(t)/M/\infty$ simulation presented here simulates 24 h of parking lot operation and treats each 24-h period as an independent replication starting with an empty garage. This only makes sense if the garage is emptied each day, for instance, if the mall closes at night. Is the assumed arrival rate $\lambda(t)$ appropriate for a mall that closes at night?
2. Suppose that the parking lot serves a facility that is actually in operation 24 h a day, 7 days per week (i.e., all the time). How should the simulation be initialized, and how long should the run length be in this case?
3. How could the simulation be initialized so that there are 100 cars in the parking lot at time 0?
4. When this example was introduced in Sect. 3.1, it was suggested that we size the garage based on the (Poisson) distribution of the number of cars in the garage at the point in time when the mean number in the garage was maximum. Is that what we did, empirically, here? If not, how is the quantity we estimated by simulation related to the suggestion in Sect. 3.1 (for instance, will the simulation tend to suggest a bigger or smaller garage than the analytical solution in Sect. 3.1?).
5. One reason that this simulation executes quite slowly when $\lambda(t) = 1000 + 100\sin(\pi t/12)$ is that the thinning method we used is very inefficient (lots of possible arrivals are rejected). Speculate about ways to make it faster.
6. For stochastic processes experts, another reason that the simulation is slow when $\lambda(t) = 1000 + 100\sin(\pi t/12)$ is that there can be 1000 or more pending departure events on `Calendar` at any time, which means that scheduling a new event in chronological order involves a slow search. However, it is possible to exploit the memoryless property of the exponential distribution of parking time to create an equivalent simulation that has only two pending events (the next car arrival and the next car departure) at any point in time. Describe how to do this.

4.3 Simulating the *M/G/*1 Queue

Here, we consider the hospital example of Sect. 3.2, a queueing system with Poisson arrival process, some (not yet specified) service-time distribution, and a single server (either a receptionist or an electronic kiosk). In other words, here we have an $M/G/1$ queue. Patient waiting time, specifically the long-run average waiting time, is the key performance measure.

In this section, we will describe both recursion-based and event-based simulations of this queue, starting with the recursion. The recursion-based simulation uses Lindley's equation (3.3), which avoids the need for an event-based simulation, but this approach is limited in the ability to track quantities of interest. For example, Lindley's equation does not provide a way to track the time-average number of entities in the system.

*4.3.1 Lindley Simulation of the M/G/*1 *Queue*

Lindley's equation (3.3) is

$$
\begin{aligned}
Y_0 &= 0 \qquad X_0 = 0 \\
Y_i &= \max\{0, Y_{i-1} + X_{i-1} - A_i\},\ i = 1, 2, \ldots,
\end{aligned}
$$

where Y_i is the ith customer's waiting time, X_i is that customer's service time, and A_i is the interarrival time between customers $i-1$ and i.

Suppose that the mean time between arrivals is 1 min, with the distribution being exponential, and the mean time to use the kiosk is 0.8 min (48 s), with the distribution being an Erlang-3. An Erlang-p is the sum of p i.i.d. exponentially distributed random variables, so an Erlang-3 with mean 0.8 is the sum of three exponentially distributed random variables each with mean 0.8/3.

Recall from Sect. 3.2 that the waiting-time random variables $Y_1, Y_2, \ldots$ converge in distribution to a random variable Y. To summarize the long-run performance of the queueing system, we are interested in $\mu = \mathrm{E}(Y)$. We know that $\bar{Y}(m) = m^{-1}\sum_{i=1}^{m} Y_i$ converges with probability 1 to μ as the number of simulated customers m goes to infinity.

Although it seems reasonable to make a very long simulation run by choosing a very large m and using $\bar{Y}(m)$ to estimate μ, we do not use this approach due to the following issue. Waiting times early in a simulation run pull $\bar{Y}(m)$ down. This behavior occurs because at the beginning of a simulation run, the queue starts empty, and waiting times early in the run tend to be smaller than μ. The impact of early waiting times with a downward bias becomes negligible as $m \to \infty$, but in practice whatever number m of customers we simulate is finite and the impact of early waiting times does affect $\bar{Y}(m)$.

To reduce this impact of early waiting times, we use a warm-up period. During a simulation run, we let the simulation generate some waiting times (say d of them) before starting to actually include them in our average. We still make m large, but our average only includes the last $m-d$ waiting times. That is, we use the truncated average as our estimator:

```
import SimRNG
import pandas
import numpy

ZRNG = SimRNG.InitializeRNSeed()

WaitTimeAvg = []

# Parameters to specify
# Units are in minutes
MeanTBA = 1.0 # mean time-between-arrivals
MeanST = 0.8 # mean service time
Phases = 3 # number of Erlang distribution phases
m = 55000 # number of customers to simulate
d = 5000 # number of customers in warm-up period

# Number of simulation experiment replications
NumReps = 10

for Rep in range(NumReps):

    Y = 0
    SumY = 0

    # Warm-up period -- these waiting times
    #   are not saved or recorded
    for i in range(0,d):
        A = SimRNG.Expon(MeanTBA, 1)
        X = SimRNG.Erlang(Phases, MeanST, 2)
        Y = max(0, Y + X - A)

    # End of warm-up period -- these waiting times
    #   are kept and used for estimation
    for i in range(d,m):
        A = SimRNG.Expon(MeanTBA, 1)
        X = SimRNG.Erlang(Phases, MeanST, 2)
        Y = max(0, Y + X - A)
        SumY = SumY + Y

    WaitTimeAvg.append(SumY/(float(m-d)))

# Create dataframe for replication statistics
#   and write to CSV file
WaitTimeAvg = pandas.DataFrame({"WaitTimeAvg": WaitTimeAvg})
WaitTimeAvg.to_csv("WaitTimeAvg.csv", sep=",")

print("Means")
print(WaitTimeAvg.mean())
```

Fig. 4.11 Simulation of the M/G/1 queue with Lindley's equation

Table 4.1 Ten replications of the $M/G/1$ queue using Lindley's equation

Replication	$\bar{Y}(55{,}000, 5000)$
1	2.191902442
2	2.291913404
3	2.147858324
4	2.114346960
5	2.031447995
6	2.110924602
7	2.132711743
8	2.180662859
9	2.139610760
10	2.146212039
Average	2.148759113
std. dev.	0.066748617

$$\bar{Y}(d,m) = \frac{1}{m-d} \sum_{i=d+1}^{m} Y_i. \tag{4.1}$$

In addition, rather than only conduct a single run of m customers, we instead make n replications, yielding n i.i.d. averages $\bar{Y}_1(d,m), \bar{Y}_2(d,m), \ldots, \bar{Y}_n(d,m)$ to which we can apply standard statistical analysis. This avoids the need to directly estimate the asymptotic variance γ^2, a topic we defer to later chapters.

Figure 4.11 shows a PythonSim simulation of the $M/G/1$ queue using Lindley's equation. We run $n = 10$ simulation replications, and in each run we simulate $m = 55{,}000$ customers and discard the data from first $d = 5000$ of them. The estimates of average waiting time from each replication are displayed in Table 4.1.

Notice that the average waiting time is a bit over 2 min and that Python, like all programming languages, displays a very large number of output digits. How many are really meaningful? A confidence interval is one way to provide an answer.

Since the across-replication averages are i.i.d., and since each across-replication average is itself the within-replication average of a large number of individual waiting times (50,000 to be exact), the assumption of independent, normally distributed output data is reasonable. This assumption justifies a t-distribution confidence interval on μ. The key ingredient is $t_{1-\alpha/2,n-1}$, the $1-\alpha/2$ quantile of the t distribution with $n-1$ degrees of freedom. If we want a 95% confidence interval, then $1-\alpha/2 = 0.975$, and our degrees of freedom are $10-1=9$. Since $t_{0.975,9} = 2.26$, we get $2.148759113 \pm (2.26)(0.066748617)/\sqrt{10}$ or $2.148759113 \pm 0.047703552$. This implies that we can claim with high confidence that μ is around 2.1 min, or we could give a little more complete information as 2.14 ± 0.05 min. Any additional digits are statistically meaningless.

Is an average of 2 min too long to wait? To actually answer that question would require some estimate of the corresponding wait to see the receptionist, either from observational data or a simulation of the current system. Statistical comparison of alternatives is a topic of Chap. 9.

4.3.2 Event-Based Simulation of the M/G/1 Queue

The event-driven simulation program consists of initialization and parameter specifications (Fig. 4.12), event routines (Fig. 4.13), a main program loop (Fig. 4.14), and output generation (Fig. 4.15).

Four PythonSim class objects and one function are illustrated by this model: `Entity`, `FIFOQueue`, `Resource`, `DTStat` and `ClearStats`. In this simulation, the `Entity` objects represent patients. We create an instance of `FIFOQueue` called `Queue` to represent the patients waiting to use the kiosk. We use a `Resource` object to represent the kiosk.

We are interested in long-run average patient waiting time, so we create a `DTStat` instance called `WaitTime` to keep track of the average of all observed waiting times within a simulation replication. `FIFOQueue` objects automatically create a `CTStat` instance that keeps track of the average number in the queue. We can also use `Queue.NumQueue()` to access the current number in the queue at any point in time. When called at the end of a replication, `Queue.NumQueue()` is the number of patients still in the queue at the end of the simulation. `Resource` objects automatically create a `CTStat` instance that keeps track of the average number of busy resource units, and in our setting this statistic is equivalent to the average kiosk utilization, i.e., the average proportion of time the kiosk is in use. Initialization of these statistics as well as parameter designation is shown in Fig. 4.12.

Next, we discuss arrival and end-of-service events, which are the two types of recurring events that drive the simulation, as shown in Fig. 4.13. `Arrival()` represents an arrival event and creates a new patient instance and adds this new patient to the queue. If the kiosk is not being used, then there are no other patients waiting either, so the patient who has just arrived can immediately begin using the kiosk. When a new patient begins service on a kiosk, we do the following: seize the server, remove the patient starting service from the queue, record that patient's waiting time, and schedule that patient's corresponding end-of-service event. Additionally, the next arrival event is scheduled. `EndOfService()` represents an end-of-service event. When called, we free the server so that it is no longer busy. If the queue is nonempty, the first patient in the queue begins service at the kiosk. Just as in the `Arrival()` routine, when this new patient begins service on a kiosk, we seize the server, remove the patient starting service from the queue, record the observed waiting time, and schedule the next end-of-service event.

Notice that `Queue.Add(Patient)` adds `Patient` to the end of `Queue`, which preserves order and a first-in-first-out policy. `Queue.Remove()` both removes *and* returns the `SimClasses.Entity()` instance at the front of `Queue`. Therefore, `NextPatient = Queue.Remove()` captures this instance and allows us to access attributes such as `NextPatient.CreateTime`, which in this example is the arrival time (in simulation clock time) of that patient. The time difference between a patient's service start time and arrival time is that patient's waiting time, which is why we log `WaitTime.Record(SimClasses.Clock - NextPatient.CreateTime)`.

Before a `Resource` object can be used, its capacity (the number of identical units) must be set. This is accomplished by using the object's `SetUnit` method, `Server.SetUnits(1)`. If, for instance, there were three identical kiosks, then this statement would be `Server.SetUnits(3)`. To make one (or more) units of a `Resource` busy, the `Seize` method is employed, while idling a `Resource` is accomplished via the `Free` method.

In addition to arrival and end-of-service events, we need two other types of events, each of which only occurs once. We require an event corresponding to the end of a "warm-up period" and an end of simulation event to stop the simulation run. The warm-up period is the time that a simulation replication runs before we collect results. At the end of the warm-up period, we clear statistics. The approach here is analogous to the truncated average approach of Sect. 4.3.1.

Because we are interested in long-run performance, we must decide on when the end of the warm-up period and when the end of the simulation run occur. This decision is not a trivial one, because using a warm-up period that is too brief may lead to biased results and simulating a replication over a time horizon that is too short cannot generate results that capture long-run behavior. Furthermore, when using Lindley's equation (3.3) and the estimator $\bar{Y}(d,m)$ (4.1), it is natural to work in units of "number of patients to simulate." In other words, d and m are patient counts. But in event-driven simulations with more complicated logic and more kinds of outputs, it is far more common to work in units of simulation clock time, which is also ideal for continuous-time statistics like the time-average number in queue. Whereas we discard data after a certain *count* of d customers when using Lindley's equation, we discard data after a certain *time* when using an event-driven simulation. Similarly, we stop a simulation replication at a stopping time chosen to be long enough to obtain good estimates for all desired outputs.

We schedule two events: a statistics clearing event "ClearIt" that calls `ClearStats` at time $t_1 = 5000$ min in the simulation run and an "EndSimulation" event at time $t_2 = 55{,}000$ min. Because the arrival rate is on average 1 per minute, t_1 and t_2 *approximately* correspond to the simulation time needed to observe 5000 and 55,000 patients, respectively; however, the actual number of patients is random and varies from replication to replication.

We define an estimator that is a function of t_1 and t_2 rather than patient counts:

$$\bar{Y}'(t_1,t_2) = \frac{1}{N(t_2) - N(t_1)} \sum_{i=N(t_1)}^{N(t_2)} Y_i, \tag{4.2}$$

where Y_i is the ith patient's waiting time, $0 < t_1 < t_2$, and

$$N(t) = \left\{ \max n : \sum_{i=1}^{n} Y_i \leq t \right\}$$

is the number of patients who have arrived and started service (finished their waiting times) by simulation time t. In other words, $\bar{Y}'(t_1,t_2)$ is a replication average of all waiting times recorded between times t_1 and t_2.

Clearly, $\bar{Y}(d,m)$, which is based on count, has different statistical properties than $\bar{Y}'(t_1,t_2)$, which is based on time. But both are good estimators of μ provided that their arguments are large enough and fixed, meaning that they do not depend on the data.

For this event-based simulation, it is easy to record and report a number of statistics. `FIFOQueue`, `Resource` and `DTStat` objects all have `Mean` methods for reporting average values. At the end of each replication, the values of `WaitTime.Mean()`, `Queue.Mean()`, `Queue.NumQueue()`, and `Server.Mean()` are appended to respective lists `WaitTimeAvg`, `QueueLengthAvg`, `QueueLengthAtEnd`, and `NumBusyServersAvg`. See Figs. 4.12 and 4.14 for details. The ith element of each list holds the statistic result from the ith simulation replication, and this format is conducive to writing the experiment data to a .csv file, as shown in Fig. 4.15.

Table 4.2 shows the results from ten replications, along with the overall averages and 95% confidence interval halfwidths. Again, there are meaningless digits, but the confidence intervals can be used to prune them. For instance, for average waiting time, we could report 2.13 ± 0.04 min.

4.3.3 Issues and Extensions

1. In what situations does it make more sense to compare the simulated kiosk system to simulated data from the current receptionist system rather than real data from the current receptionist system?
2. It is clear that if all we are interested in is mean waiting time, defined either as time until service begins or as the total time including service, the Lindley approach is superior (since it is clearly faster, and we can always add in the mean service time to the Lindley estimate). However, if we are interested in the distribution of total waiting time, then adding in the mean service time does not work. How can the Lindley recursion be modified to simulate total waiting times?
3. How can the event-based simulation be modified so that it also records the total time in system (the time waiting for service plus the service time)?
4. How can the event-based simulation be modified to clear statistics after exactly 5000 patients and to stop at exactly 55,000 patients?

```
# PythonSim imports
import SimClasses
import SimFunctions
import SimRNG

# Python package imports
import math
import pandas

# Initialization
SimClasses.Clock = 0
ZSimRNG = SimRNG.InitializeRNSeed()
Calendar = SimClasses.EventCalendar()
Queue = SimClasses.FIFOQueue()
WaitTime = SimClasses.DTStat()
Server = SimClasses.Resource()

# Example-specific statistics
# WaitTimeAvg: list of average patient waiting time
#   in each replication
# QueueLengthAvg: list of average queue length
#   in each replication
# QueueLengthAt24Hrs: list of queue length at simulation end
#   in each replication
# NumBusyServersAvg: list of time-averaged number
#   of busy servers in each replication
WaitTimeAvg = []
QueueLengthAvg = []
QueueLengthAtEnd = []
NumBusyServersAvg = []

# Parameters to specify
# Units are in minutes
Server.SetUnits(1) # set number of servers to 1
MeanTBA = 1.0 # mean time-between-arrivals
MeanST = 0.8 # mean service time
Phases = 3 # number of Erlang distribution phases
RunLength = 55000.0 # run length in minutes
WarmUp = 5000.0 # warm-up period length in minutes

# Number of simulation experiment replications
NumReps = 10
```

Fig. 4.12 Initializing the hospital simulation

```
def Arrival():
    '''
    Called when a new patient arrives
    Creates a new patient instance corresponding to
        arriving patient and adds this to the queue
    If server (kiosk) is free, then new patient
        starts service and their waiting time
        is recorded in DTStat WaitTime
    Schedules the next arrival event
    '''

    Patient = SimClasses.Entity()
    Queue.Add(Patient)

    # If kiosk idle, service begins on new patient
    if Server.CurrentNumBusy == 0:

        Server.Seize(1)
        NextPatient = Queue.Remove()
        WaitTime.Record(SimClasses.Clock
            - NextPatient.CreateTime)
        SimFunctions.Schedule(Calendar,
            "EndOfService",SimRNG.Erlang(Phases,MeanST,2))

    SimFunctions.Schedule(Calendar,
        "Arrival",SimRNG.Expon(MeanTBA, 1))

def EndOfService():
    '''
    Called when a patient finishes service
    If there are other patients in the queue,
        then patient at the front starts service,
        their end of service event is scheduled,
        and their waiting time is recorded
    Otherwise, the server (kiosk) becomes idle
    '''

    Server.Free(1)

    # Next patient in line starts using kiosk
    if Queue.NumQueue() > 0:

        Server.Seize(1)
        NextPatient = Queue.Remove()
        WaitTime.Record(SimClasses.Clock
            - NextPatient.CreateTime)
        SimFunctions.Schedule(Calendar,
            "EndOfService",SimRNG.Erlang(Phases,MeanST,2))
```

Fig. 4.13 Event routines for the hospital simulation

```
for reps in range(0,NumReps):

    # Initialize simulation objects and
    #   schedule first event, end-of-simulation event,
    #   and event to clear statistics after warm-up period
    SimFunctions.SimFunctionsInit(Calendar)
    SimFunctions.Schedule(Calendar,
        "Arrival",SimRNG.Expon(MeanTBA, 1))
    SimFunctions.Schedule(Calendar,"EndSimulation",RunLength)
    SimFunctions.Schedule(Calendar,"ClearIt",WarmUp)

    # Main simulation loop
    while Calendar.N() > 0:
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
        if NextEvent.EventType == "Arrival":
            Arrival()
        elif NextEvent.EventType == "EndOfService":
            EndOfService()
        elif NextEvent.EventType == "ClearIt":
            SimFunctions.ClearStats()
        elif NextEvent.EventType == "EndSimulation":
            break

    # Add replication statistics to respective lists
    WaitTimeAvg.append(WaitTime.Mean())
    QueueLengthAvg.append(Queue.Mean())
    QueueLengthAtEnd.append(Queue.NumQueue())
    NumBusyServersAvg.append(Server.Mean())
```

Fig. 4.14 Main program for the hospital simulation

```
# Create dataframe for replication statistics
# and write to csv file
output = pandas.DataFrame(
    {"WaitTimeAvg": WaitTimeAvg,
    "QueueLengthAvg": QueueLengthAvg,
    "QueueLengthAtEnd" : QueueLengthAtEnd,
    "NumBusyServersAvg": NumBusyServersAvg})
output.to_csv("MG1_output.csv", sep=",")

print("Means")
print(output.mean())

print("Standard Deviation")
print(numpy.sqrt(output.var(ddof = 1)))

print("95% CI Half-Width")
print(1.96*numpy.sqrt(output.var(ddof = 1)/len(output)))
```

Fig. 4.15 Output for the hospital simulation

Table 4.2 Ten replications of the $M/G/1$ queue using event-based simulation

Replication	Wait Avg	Queue Avg	Remaining	Utilization
1	2.195532	2.210769	0	0.806655
2	2.302218	2.320497	6	0.807273
3	2.150942	2.149009	1	0.799339
4	2.053335	2.051967	0	0.794694
5	2.107930	2.101822	1	0.798616
6	2.093995	2.094918	2	0.801967
7	2.108793	2.102555	0	0.798657
8	2.117575	2.113171	3	0.795435
9	2.123434	2.124378	0	0.799157
10	2.070940	2.063029	0	0.799144
Average	2.132469	2.133211	1.300000	0.800093
std. dev.	0.071591	0.079399	1.946507	0.004157
±95% CI	0.044372	0.049212	1.206457	0.002577

5. The experiment design method illustrated in the event-based simulation is often called the "replication–deletion" method. If we only had time to generate 500,000 waiting times, what issues should be considered in deciding the values of n (replications), m (run length), and d (deletion amount)? Notice that we must have $nm = 500{,}000$, and only $n(m-d)$ observations will be used for estimating the mean μ.
6. An argument against summarizing system performance by long-run measures is that no system stays unchanged forever (for instance there could be staff changes, construction, or emergencies), so a measure like μ is not a reflection of reality. The counter-argument is that it is difficult, if not impossible, to model all of the detailed changes that occur over any time horizon (even the time-dependent arrival process in the $M(t)/M/\infty$ simulation is difficult to estimate in practice), so long-run performance at least provides an understandable summary measure. ("If our process stayed the same, then over the long run…") Also, it is often mathematically easier to obtain long-run measures than it is to estimate them by simulation (since simulations have to stop). Considering these issues, what sort of analysis makes the most sense for the hospital problem?

4.4 Simulating the Stochastic Activity Network

Here, we consider the construction example of Sect. 3.4, which is represented as a stochastic activity network (SAN). Recall that the time to complete the project, Y, can be represented as

$$Y = \max\{X_1 + X_4, X_1 + X_3 + X_5, X_2 + X_5\},$$

where X_i is the duration of the ith activity. This simple representation requires that we enumerate all paths through the SAN, so that the project completion time is the longest of these paths. Path enumeration itself can be time consuming, and this approach does not easily generalize to projects that have resources shared between activities, for instance. Therefore, we also present a discrete-event representation, which is more complicated, but also more general.

4.4.1 Maximum Path Simulation of the SAN

Figure 4.16 shows an implementation of the algorithm displayed in Sect. 3.4 and repeated here:

```
1. set $s = 0$
2. repeat $n$ times:

    a. generate $X_1, X_2, \ldots, X_5$
    b. set $Y = \max\{X_1 + X_4, X_1 + X_3 + X_5, X_2 + X_5\}$
    c. if $Y > t_p$ then set $s = s + 1$

3. estimate $\theta$ by $\widehat{\theta} = s/n$
```

Since $\Pr\{Y \leq t_p\}$ is known for this example (see Eq. (3.12)), the true $\theta = \Pr\{Y > t_p\} = 0.165329707$ when $t_p = 5$ is also computed by the program so that we can compare it to the simulation estimate. Of course, in a practical problem, we would not know the answer, and we would be wasting our time simulating it if we did. Notice that all of the digits in this probability are correct—assuming that the numerical functions in Python did their job—although certainly not practically useful.

The simulation estimate turns out to be $\widehat{\theta} = 0.164$, and the Python code for computing this estimate is displayed in Fig. 4.16. A nice feature of a probability estimate that is based on i.i.d. outputs is that an estimate of its standard errors is easily computed:

$$\widehat{\text{se}} = \sqrt{\frac{\widehat{\theta}(1-\widehat{\theta})}{n}}.$$

Thus, $\widehat{\text{se}}$ is approximately 0.011, and the simulation has done its job since the true value θ is well within $\pm 1.96\,\widehat{\text{se}}$ of $\widehat{\theta}$. This is a reminder that simulations do not deliver *the answer*, such as Eq. (3.12), but do provide the capability to estimate the simulation error and to reduce that error to an acceptable level by increasing the simulation effort (the number of replications).

```
# imports
import SimRNG
import math

ZRNG = SimRNG.InitializeRNSeed()

NumReps = 10000

# Counter for number of times Y > tp
counter = 0

# Desired completion time for the project
tp = 5

# Main simulation loop
for rep in range(0,NumReps):
    X = []
    for i in range(0,5):
        X.append(SimRNG.Expon(1.0,7))
    Y = max(X[0] + X[3], X[0] + X[2] + X[4], X[1] + X[4])

    if Y > tp:
        counter = counter + 1

# Compute closed-form solution for P{Y < tp}
Theta = 1- ((tp ** 2 / 2.0 - 3 * tp - 3)
    * math.exp(-2 * tp)
    + (-tp ** 2 / 2.0 - 3 * tp + 3)
    * math.exp(-tp) + 1 - math.exp(-3 * tp))

print("Estimated P{Y < tp}")
print(counter/NumReps)

print("Closed form P{Y < tp}")
print(Theta)
```

Fig. 4.16 Simulation of the SAN as the maximum path through the network

4.4.2 Discrete-Event Simulation of the SAN

As usual, we organize our event-driven simulation into subsections of initialization and parameter specifications (Figs. 4.17 and 4.18), definition of a "milestone" event (Fig. 4.19), the main simulation loop (Fig. 4.20), and output generation (Fig. 4.21).

This example uses more advanced PythonSim constructs than any other part of the book and may be skipped without loss of continuity. Specifically, we discuss `Activity` and `Node` objects and the use of `SchedulePlus` to assign objects to the `WhichObject` attribute of `EventNotice` instances.

```
# PythonSim imports
import SimClasses
import SimFunctions
import SimRNG

# Python package imports
import pandas

# Initialization
SimClasses.Clock = 0.0
ZSimRNG = SimRNG.InitializeRNSeed()
Calendar = SimClasses.EventCalendar()

# Number of simulation replications
NumReps = 10000

# Stores each replication’s completion time
CompletionTime = []

a = SimClasses.Node()
b = SimClasses.Node()
c = SimClasses.Node()
d = SimClasses.Node()

X1 = SimClasses.Activity()
X2 = SimClasses.Activity()
X3 = SimClasses.Activity()
X4 = SimClasses.Activity()
X5 = SimClasses.Activity()

X1.Destination = b
X2.Destination = c
X3.Destination = c
X4.Destination = d
X5.Destination = d
```

Fig. 4.17 Initializing the discrete-event SAN simulation

First, we discuss `Activity` and `Node` objects and how they are used to capture the structure of a stochastic activity network. As shown in Fig. 4.17, we initialize one `Node` instance for each node in the SAN, creating `a,b,c`, and `d`. We initialize one `Activity` instance for each activity and assign each activity’s corresponding destination node to the activity’s `Destination` attribute. For example, `X1.Destination = b`.

We create a function `SANInit()` to be called at the beginning of each replication (see Fig. 4.18). In this function, we fill in each `Node` object’s `Incoming` and `Outgoing` attributes and also set each `Activity` object’s `CompletionTime` to a random variable with the correct distribution. A `Node` object’s `Incoming` attribute is a list of `Activity` objects, representing unfinished incoming activities. As these activities are completed within a simulation run, they are removed from

```
def SANInit():
    '''
    Called at beginning of every replication
    Defines example-specific graph structure
        and specifies the distribution
        for each activity duration

    For each Node instance,
        Incoming attribute is list of unfinished
            incoming activities
        Outgoing attribute is a list of unscheduled
            outgoing activities
        Activity instances are removed
            from their lists when completed
    '''

    a.Incoming = []
    a.Outgoing = [X1,X2]

    b.Incoming = [X1]
    b.Outgoing = [X3,X4]

    c.Incoming = [X2,X3]
    c.Outgoing = [X5]

    d.Incoming = [X4,X5]
    d.Outgoing = []


    X1.CompletionTime = SimRNG.Expon(1,1)
    X2.CompletionTime = SimRNG.Expon(1,1)
    X3.CompletionTime = SimRNG.Expon(1,1)
    X4.CompletionTime = SimRNG.Expon(1,1)
    X5.CompletionTime = SimRNG.Expon(1,1)
```

Fig. 4.18 Initializing the discrete-event SAN simulation, continued

the list. The `Outgoing` attribute is also a list of `Activity` objects and represents outgoing activities that are not yet scheduled on the event calendar. Only after all incoming activities into a node are finished can the node's outgoing activities commence, and once they are scheduled, they are removed from `Outgoing`. We wrap these aforementioned assignments in `SanInit()` because we need each node's incoming and outgoing lists to reset at the beginning of each simulation replication, and we need new random variables for each activity's random completion times with each simulation replication.

Next, we discuss the only type of event we need, called "event milestones." Event milestones are described in Fig. 4.19. As noted in Sect. 3.4, we can think of the completion of a project activity as an event. When all of the inbound activities $\mathscr{I}(j)$

to a node j are completed, then the outbound activities $i \in \mathscr{O}(j)$ are scheduled, with the destination milestone of activity i being $\mathscr{D}(i)$. A brief description of event milestone logic is shown below.

```
**event milestone** (activity $\ell$ inbound to node $j$)
$\mathscr{I}(j) = \mathscr{I}(j) - \ell$
if $\mathscr{I}(j) = \emptyset$ then
    for each activity $i \in \mathscr{O}(j)$
        schedule milestone(activity $i$ inbound to node $\mathscr{D}(i)$ at $X_i$ time units later)
end if
```

The event-driven simulation approach using milestones avoids enumeration of all of the paths through the SAN. We only have to keep track of the sets $\mathscr{I}(j)$ and $\mathscr{O}(j)$ for each node j and the destinations $\mathscr{D}(i)$ for each activity i, which we need to specify any way to define the project itself, regardless of which simulation approach we use. We take care of housekeeping for these sets and destinations using our `Activity` and `Node` objects. One key takeaway from the event-driven SAN simulation is that object-oriented programming can capture network structures in a simple way.

The main simulation loop is displayed in Fig. 4.20. Each simulation is kick-started by scheduling the first milestone with the function `Milestone(None,a)`. We input `None` as the parameter for `ActivityIn` because `a` is the first node and has no incoming activities, and the first events on the event calendar are `a`'s outgoing activities.

The SAN example introduces the function `SchedulePlus`, which is identical to `Schedule` except for the addition of a fourth parameter called `TheObject`. The argument for `TheObject` becomes an additional attribute `WhichObject` of an `EventNotice` instance, which is then added to the event calendar in the usual way. Using `SimFunctions.SchedulePlus(Calendar, ''Milestone'', NextActivity.CompletionTime, NextActivity)` (Fig. 4.19) is important because of the special structure of a SAN—we need to know *which* activity is completed whenever a milestone event occurs. At each event, we set `CurrentActivity = NextEvent.WhichObject`, which determines the next milestone `Milestone(CurrentActivity, CurrentActivity.Destination)` (Fig. 4.20). Because we need the inbound activity and the target node, and since this information is needed when a milestone event is executed, not when it is scheduled, we need to store this information with the event notice. Using `SchedulePlus` makes it possible to program a single event routine to handle many simulation events that are conceptually distinct, by passing event-specific information to the event routine.

The SAN project ends when all activities have been completed, so rather than create an event for a simulation end time, we simply end each simulation run when there are no longer any events on the event calendar. As shown in Fig. 4.20, `Calendar.N` returns the number of events currently on the event calendar, and the simulation replication ends when this quantity is `0`. At simulation termination, the simulation clock time `SimClasses.Clock` is the project completion time. We

```
def Milestone(ActivityIn, DestinationNode):
    '''
    Removes ActivityIn from DestinationNode's
        Incoming list
    If all incoming activities of DestinationNode
        have finished, begin all outgoing activities
        and schedule completion times

    Input:
        ActivityIn: Activity object
        DestinationNode: Node object that is the
            destination of ActivityIn
    '''

    if ActivityIn in DestinationNode.Incoming:
        DestinationNode.Incoming.remove(ActivityIn)

    if len(DestinationNode.Incoming) == 0:
        while len(DestinationNode.Outgoing) > 0:
            NextActivity = DestinationNode.Outgoing.pop()
            SimFunctions.SchedulePlus(Calendar,"Milestone",
                NextActivity.CompletionTime, NextActivity)
```

Fig. 4.19 Milestone event for the discrete-event SAN simulation

append this quantity to a list `CompletionTime`, which stores completion times from all replications, and Fig. 4.21 provides a basic script for simple data analysis.

We can estimate $\Pr\{Y > t_p\}$, where Y is project completion time, for any given value of t_p by counting how many entries of `CompletionTime` are greater than t_p. Figure 4.22 shows the empirical cdf of 1000 project completion times, which is the simulation estimate of Eq. (3.12).

4.4.3 Issues and Extensions

1. In real projects, there are not only activities but also limited and often shared resources that are needed to complete the activities. Furthermore, there may be specific resource allocation rules when multiple activities contend for the same resource. How might this be modeled in PythonSim?
2. Time to complete the project is an important overall measure, but at the planning stage it may be more important to discover which activities or resources are the most critical to on-time completion of the project. What additional output measures might be useful for deciding which activities are “critical”?

```
for rep in range(0,NumReps):

    # Reset incoming and outgoing activities
    #   for each node and reset other simulation objects
    SANInit()
    SimFunctions.SimFunctionsInit(Calendar)

    # Set up first milestone
    # This also schedules initial calendar events
    #   since the starting node has no incoming
    #   activities
    Milestone(None,a)

    # Main simulation loop
    while Calendar.N() > 0:
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime
        CurrentActivity = NextEvent.WhichObject
        Milestone(CurrentActivity,CurrentActivity.Destination)

    CompletionTime.append(SimClasses.Clock)
```

Fig. 4.20 Main program for the discrete-event SAN simulation

```
CompletionTime = pandas.DataFrame(
    {"Completion Time": CompletionTime})
CompletionTime.to_csv("CompletionTime.csv", sep=",")

print("Means")
print(CompletionTime.mean())
```

Fig. 4.21 Output for the discrete-event SAN simulation

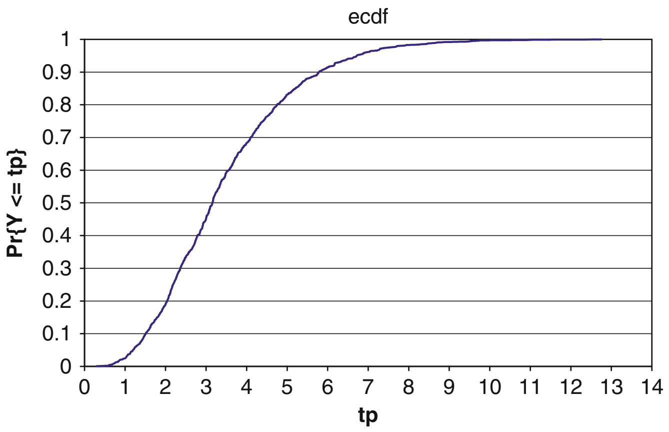

Fig. 4.22 Empirical cdf of the project completion times

4.5 Simulating the Asian Option

Here, we consider estimating the value of an Asian option

$$\nu = \mathrm{E}\left[\mathrm{e}^{-rT}\left(\bar{X}(T) - K\right)^{+}\right]$$

as described in Sect. 3.5. This estimation problem is an example in which a discrete-event simulation approach is actually unnecessary and even undesirable due to overhead. Instead of using an event calendar, we use a simple loop for numerical computation.

The maturity is $T = 1$ year, the risk-free interest rate is $r = 0.05$ and the strike price is $K = \$55$. The underlying asset has an initial value of $X(0) = \$50$ and the volatility is $\sigma^2 = (0.3)^2$.

Recall that the key quantity is

$$\bar{X}(T) = \frac{1}{T}\int_0^T X(t)\,\mathrm{d}t,$$

the time average of a continuous-time, continuous-state geometric Brownian motion process, which we cannot truly simulate on a digital computer. Thus, we approximate $\bar{X}(T)$ it by dividing the interval $[0,T]$ into m steps $t_1, t_2, \ldots, t_{m-1}, t_m$ and using the discrete approximation

$$\widehat{\bar{X}(T)} = \frac{1}{m}\sum_{i=1}^{m} X(t_i).$$

This makes simulation possible, since

$$X(t_{i+1}) = X(t_i)\exp\left\{\left(r - \frac{1}{2}\sigma^2\right)(t_{i+1} - t_i) + \sigma\sqrt{t_{i+1} - t_i}\,Z_{i+1}\right\}$$

for any increasing sequence of times $\{t_0, t_1, \ldots, t_m\}$, where $Z_1, Z_2, \ldots, Z_m$ are i.i.d. $\mathrm{N}(0,1)$.

Figure 4.23 displays code for estimating ν. Here we use $m = 32$ steps in the approximation, and we use equally spaced steps, so that $t_{i+1} - t_i = T/m$ for all $i = 0, 1, 2, \ldots, m-1$. We run 10,000 replications to estimate ν. Discrete-event structure would slow execution without any obvious benefit, so a simple loop is used to advance time. The value of the option from each replication is written to a .csv file for the post-simulation analysis.

The estimated value of ν is \$2.20 with a relative error of just over 2% (recall that the relative error is the standard error divided by the mean). As the histogram in Fig. 4.24 shows, the option is frequently worthless (approximately 68% of the time), but the average payoff, conditional on the payoff being positive, is approximately \$6.95.

```
# Imports
import SimRNG
import math
import pandas

# Initialization
ZRNG = SimRNG.InitializeRNSeed()

# Option parameters
Maturity = 1.0
InterestRate = 0.05
StrikePrice = 55.0
InitialValue = 50.0
Sigma = 0.3

# Number of approximation steps
Steps = 32

# Number of simulation replications
NumReps = 10000

# Distance between successive timepoints (t_{i+1} - t_i)
Interval = Maturity / Steps

# List to hold replication output
TotalValue = []

# Main simulation loop
for i in range(0,NumReps):

    Sum = 0.0
    X = InitialValue

    for j in range(0,Steps):
        Z = SimRNG.Normal(0,1,12)
        X = X * math.exp((InterestRate - (Sigma **2 / 2))
            * Interval + Sigma * math.sqrt(Interval) * Z)
        Sum += X

    Value = (math.exp(-InterestRate * Maturity)
        * max(Sum / Steps - StrikePrice, 0))

    TotalValue.append(Value)

TotalValue = pandas.DataFrame({"Total Value": TotalValue})
TotalValue.to_csv("TotalValue.csv", sep = ",")

print("Replication Stats")
print(TotalValue)

print("Means")
print(TotalValue.mean())
```

Fig. 4.23 VBA simulation of the Asian option problem

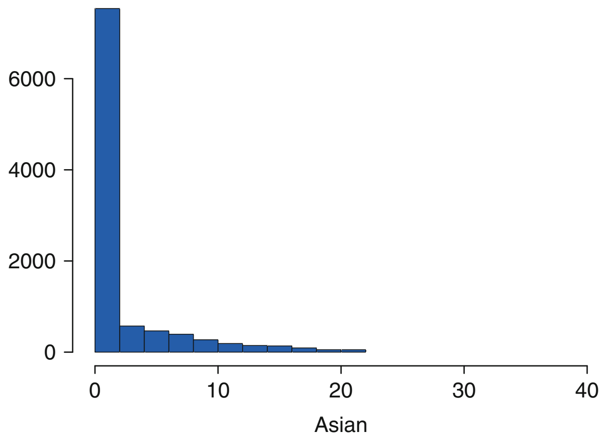

Fig. 4.24 Histogram of the realized value of the Asian option from 10,000 replications

4.6 Case Study: Service Center Simulation

This section presents a simulation case based on a project provided by a former student. While still relatively simple, it is more complex than the previous stylized examples, and an answer is not available to us without simulation. The purpose of this section is to illustrate how one might attack simulation modeling and programming for a realistic problem.

4.6.1 Fax Center Staffing: Example

A service center receives faxed orders[1] throughout the day, with the rate of arrival varying hour by hour. The arrivals are modeled by a nonstationary Poisson process with the rates shown in Table 4.3. There is a bank of fax machines dedicated to incoming faxes, so it is reasonable to treat the arrival of faxes as an unconstrained external arrival process.

A team of Entry Agents selects faxes on a first-come-first-served basis from the fax queue. Their time to process a fax is modeled as normally distributed with mean 2.5 min and standard deviation 1 min. There are two possible outcomes after the Entry Agent finishes processing a fax: either it is a simple fax and the work on it is complete, or it is not simple and it needs to go to a Specialist for further processing. Over the course of a day, approximately 20% of faxes require a Specialist. The time for a Specialist to process a fax is modeled as normally distributed with mean 4.0 min and standard deviation 1 min.

[1] A fax is like an old-school, physical email. The first fax machine was actually created in 1843, which is roughly the time Nelson finished his Ph.D.

Table 4.3 Arrival rate of faxes by hour

Time	Rate (faxes/minute)
8 a.m.–9 a.m.	4.37
9 a.m.–10 a.m.	6.24
10 a.m.–11 a.m.	5.29
11 a.m.–12 p.m.	2.97
12 p.m.–1 p.m.	2.03
1 p.m.–2 p.m.	2.79
2 p.m.–3 p.m.	2.36
3 p.m.–4 p.m.	1.04

Minimizing the number of staff minimizes cost, but certain service-level requirements must be achieved. In particular, 96% of all simple faxes should be completed within 10 min of their arrival, while 80% of faxes requiring a Specialist should also be completed (by both the Entry Agent and the Specialist) within 10 min of their arrival.

The service center is open from 8 a.m. to 4 p.m. daily, and it is possible to change the staffing level at 12 p.m. Thus, a staffing policy consists of four numbers: the number of Entry Agents and Specialists before noon and the number of Entry Agents and Specialists after noon. Any fax that starts its processing before noon completes processing by that same agent before the agent goes off duty, and faxes in the queues at the end of the day are processed before the agents leave work and therefore are not carried over to the next day.

4.6.2 Fax Center Staffing: Problem Modeling and Analysis

The first step in building any simulation model is deciding what question or questions that the model should answer. Knowing the questions helps identify the system performance measures that the simulation needs to estimate, which in turn drives the scope and level of detail in the simulation model.

The grand question for the service center is, what is the minimum number of Entry Agents and Specialists needed for both time periods to meet the service-level requirements? Therefore, the simulation must at least provide an estimate of the percentage of faxes of each type entered within 10 min, given a specific staff assignment.

Even when there seems to be a clear overall objective (e.g., minimize the staff required to achieve the service-level requirement), we often want to consider tradeoffs around that objective. For instance, if meeting the requirement requires a staff that is so large that they are frequently underutilized, or if employing the minimal staff means that the Entry Agents or Specialists frequently have to work well past the end of the day, then we might be willing to alter the service requirement a bit. Statistics on the number and the time spent by faxes in queue, and when the last fax of each day is actually completed, provide this information. Including additional measures of system performance, beyond the most critical ones, makes the simulation more useful.

Many discrete-event, stochastic simulations involve entities that dynamically flow through some sort of queueing network with constrained resources. In such simulations, identifying the entities and resources is a good place to start the model. For this service center, the faxes are the dynamic entities, while the Entry Agents and Specialists are resources. Since there is a bank of fax machines dedicated to incoming faxes, we treat the arrival of faxes as an unconstrained external arrival process, and we do not model the fax machines as a resource. But in other situations, if the fax machines are heavily utilized or if outgoing as well as incoming faxes use the same machines, considering fax machines as a resource might be a necessary model addition. In practice, a client's initial problem description might not provide all of the information necessary for modeling; follow-up questions are often needed to fully understand the system of interest.

Next, we discuss queueing process modeling. Whenever there are scarce resources, queues can form, and queueing behavior is often a critical part of the model. Queues are often first-in-first-out, with one queue for each resource, as is the case for this service center example. However, queues might have priorities, and multiple queues might be served by the same resource, or a single queue might feed multiple resources.

When the simulation involves entities flowing through a network of queues, then there are two possible types of arrivals: "external" arrivals from outside of the network and "internal" arrivals from within the network. Outside arrivals are like those we have already seen in the $M(t)/M/\infty$ and $M/G/1$ examples. Internal arrivals are departures from one queue that become arrivals to others. How these are modeled depends largely on whether the departure from one queue is an immediate arrival to the next—in which case the departure and arrival events are effectively the same thing—or whether there is some sort of transportation delay—in which case the arrival to the next queue should be scheduled as a distinct event. For the service center, the arrival of faxes to the Entry Agents is an outside arrival process, while the 20% of faxes that require a Specialist are internal arrivals from the Entry Agents to the Specialists.

Critical to experiment design is defining what constitutes a replication. Replications should be independent and identically distributed. Since the service center does not carry faxes over from 1 day to the next, a "day" defines a replication. If faxes do carry over, but all faxes are cleared weekly, then a replication might be defined by a work week. However, if there is always significant carry over from 1 day to the next, then a replication might have to be defined arbitrarily.

The work day at the service center is 8 h; however, the staff does not leave until all faxes that arrive before 4 p.m. are processed. If we define a replication to be exactly 8 h, then we could be fooled by a staffing policy that allows a large queue of faxes to build up toward the end of the day, since the entry of those faxes would not be included in our statistics. To model a replication that ends when there is no additional work remaining, we cut off fax arrivals at 4 p.m. and then end the simulation when the event calendar is empty. This works because idle Entry Agents and Specialists always take a fax from their queue if one is available.

4.6.3 Fax Center Staffing: PythonSim and Output Analysis

Here we discuss the `PythonSim` code and logic for a discrete-event simulation of the fax center example. We implement 15 Entry Agents in the AM shift and 9 in the PM shift and 6 Specialists in the AM shift and 3 in the PM shift, but these decision variables can be changed as desired. We split the code into distinct parts. Figure 4.25 initializes the simulation and defines the problem-specific parameters and decision variables. Figure 4.26 completes initialization by setting up the necessary `DTStat`, `Resource`, and `FIFOQueue` objects and creating lists for example-specific statistics data from each replication. Figure 4.27 is code for the nonstationary Poisson process for fax arrivals, Figs. 4.28 and 4.29 provide routines for arrival and end-of-service events at Entry Agents and Specialists, respectively, and Fig. 4.30 provides a routine for the midday staffing change. Finally, Fig. 4.31 is the main simulation loop, and Fig. 4.32 provides some simple output generation.

Rather than go through each line or function one by one, we highlight the key elements of the simulation.

We note that the two `DTStat` statements defining `RegPropWithin10` and `SpPropWithin10` in Fig. 4.30 are used to obtain the fraction of regular and "special" faxes (faxes that require additional service at a Specialist) that are completed within the 10 minute requirement by recording a 1 for any fax that meets the requirement, and a 0 otherwise. The mean of these values is the desired fraction, illustrating how `DTStat` objects can be used for indicator variables or estimating proportions. Whenever a fax is completed, we compute its time-in-system (variable `TIS` in the code, in `EntryAgentEndOfService` and `SpecialistEndOfService` in Figs. 4.28 and 4.29, respectively). We record the time-in-system and also record whether it is less than 10 min. Notice that we can record `(TIS < 10)` for our `DTStat` objects `RegWithin10` and `SpWithin10`. The quantity `(TIS < 10)` can be added to other Booleans or integers and behaves like `1` if `True` and `0` if `False`. `RegWithin10.Sum` is the total number of regular faxes completed under 10 min, and `SpWithin10.Sum` is analogous for special faxes.

Next we discuss the external arrival process, as well as the logic for cutting off arrivals after 4 p.m.. The function `NSPP` in Fig. 4.27 generates the interarrival times for faxes with the desired time-varying rate. This function is similar to the one defined in Fig. 4.5 in the $M(t)/M/\infty$ example in Sect. 4.2. The key difference is that the fax center example has piecewise constant arrival rate that changes every hour and is specified for 8 h, while the $M(t)/M/\infty$ example has a continuous function defining its arrival rate for all times $t \geq 0$. Here, if `NSPP` is computing a possible arrival time that would occur after 4 p.m., it uses the last hour's arrival rate (the rate from 3 p.m. to 4 p.m.). The function `NSPP` always returns an inter-

```
# PythonSim imports
import SimFunctions
import SimRNG
import SimClasses

# Python package imports
import math
import pandas
import numpy

# Initialize simulation
ZRNG = SimRNG.InitializeRNSeed()
Calendar = SimClasses.EventCalendar()

# Mean and variance parameters for
#   normal distribution for service time
#   for regular and special faxes
RegMean = 2.5
RegVar = 1.0
SpMean = 4.0
SpVar = 1.0
RunLength = 480

# Center is open from 8am-4pm (8 hours)
# 60 minutes in an hour ("period")
# ARate is arrival rate in faxes per minute
# MaxRate is used for thinning in NSPP
NPeriods = 8
PeriodLength = 60
ARate = [4.37,6.24,5.29,2.97,2.03,2.79,2.36,1.04]
MaxRate = max(ARate)

# Specify staffing schedule
NumEntryAgentsAM = 15
NumEntryAgentsPM = 9
NumSpecialistsAM = 6
NumSpecialistsPM = 3

# Number of simulation replications
NumReps = 10
```

Fig. 4.25 Initializing the service center simulation

```
# Simulation statistics trackers
RegTIS = SimClasses.DTStat()
RegWithin10 = SimClasses.DTStat()
SpTIS = SimClasses.DTStat()
SpWithin10 = SimClasses.DTStat()
EntryAgents = SimClasses.Resource()
EntryAgentsQueue = SimClasses.FIFOQueue()
Specialists = SimClasses.Resource()
SpecialistsQueue = SimClasses.FIFOQueue()

# Example-specific statistics
# RegTISAvg: list of average time-in-system (min)
#   for regular faxes in each replication
# RegPropWithin10: list of proportion of
#   regular faxes completed in <= 10min in each
#   replication
# SpTISAvg: same as RegTISAvg but for special faxes
# SpPropWithin10: same as RegPropWithin10 but
#   for special faxes
# EntryAgentsBusyAvg: list of average number of
#   busy EntryAgents units in each replication
# EntryAgentsQueueAvg: list of average queue length
#   for EntryAgents resource in each replication
# SpecialistsBusyAvg: same as EntryAgentsBusyAvg
#   but for Specialists
# SpecialistsQueueAvg: same as EntryAgentsQueueAvg
#   but for special faxes at the Specialists queue
# EndingTime: list of times at which all faxes
#   arriving before 4pm are completed
RegTISAvg = []
RegPropWithin10 = []
SpTimeAvg =[]
SpPropWithin10 = []
EntryAgentsBusyAvg = []
EntryAgentsQueueAvg = []
SpecialistsBusyAvg = []
SpecialistsQueueAvg = []
EndingTime = []
```

Fig. 4.26 Initializing continued: setting up statistics

arrival time, even if it corresponds to an arrival scheduled to occur after 4 p.m. When called, the `Arrival` function in Fig. 4.28 generates an interarrival time using `NSPP` and checks if `SimClasses.Clock + InterarrivalTime > RunLength`, where `RunLength` is set to 480, referring to the 480 min between 8 a.m. and 4 p.m. If this condition is true, the `Arrival` function terminates and an arrival is not scheduled. In fact, if this condition is true, there are no more arrival events for the rest of the simulation replication, successfully preventing arrivals af-

```
def NSPP(Stream):
    '''
    Generates next interarrival time from a nonstationary
        Poisson arrival process using thinning method
    Uses final hour's arrival rate for any possible arrivals
        that would occur after closing time

    Input:
        Stream: integer, corresponding to random number stream

    Output:
        nspp: float, positive, time until next nonstationary
            Poisson arrival
    '''

    PossibleArrivalTime = SimClasses.Clock
    Ratio = 0

    while SimRNG.Uniform(0, 1, Stream) >= Ratio:
        PossibleArrivalTime += SimRNG.Expon(1.0/MaxRate, Stream)
        i = int(min(NPeriods,
            math.ceil(PossibleArrivalTime/PeriodLength)))
        Ratio = ARate[i-1]/MaxRate

    nspp = PossibleArrivalTime - SimClasses.Clock
    return nspp
```

Fig. 4.27 NSPP function

ter 4 p.m. This occurs because there is at most one arrival event on the calendar at any time, and a new arrival event is scheduled only when an arrival event has just occurred.

Figure 4.28 contains the arrival and end-of-service events for faxes at the Entry Agents. When a fax arrives, either it starts service immediately if an Entry Agent is free, or it joins the queue for the Entry Agents. When an Entry Agent finishes service on a fax, with 20% probability, the `DepartingFax` that has just completed service is sent directly and immediately to the Specialists with the call `SpecialArrival(DepartingFax)`. We pass `DepartingFax` because we need to keep track of each fax's time-in-system. Notice that in our implementation, we do not schedule a special arrival event, because we simply call `SpecialArrival(DepartingFax)` at an Entry Agent end-of-service event if the departing fax needs a Specialist. Alternatively, we could schedule a special arrival event occurring zero time units in the future (or nonzero time units in the future if, for instance, there is a travel time delay between Entry Agents and Specialists). Either approach is correct and is a modeling choice.

Figure 4.29 contains the arrival and end-of-service events for faxes at the Specialists. These functions are analogous to their Entry Agent counterparts. The Specialists have their own `Resource` object, `FIFOQueue` object, and corresponding

```
def Arrival():
    '''
    Called when a new fax arrives

    If next arrival would occur after center has closed,
        function terminates and no next arrival is scheduled,
        otherwise, next arrival is scheduled
    If an Entry Agent is free, new fax seizes 1 unit and
        starts service, otherwise fax is added to EntryAgentsQueue
    '''

    InterarrivalTime = NSPP(1)
    if SimClasses.Clock + InterarrivalTime > RunLength:
        return

    SimFunctions.Schedule(Calendar,"Arrival",InterarrivalTime)
    Fax = SimClasses.Entity()

    if EntryAgents.CurrentNumBusy < EntryAgents.NumberOfUnits:
        EntryAgents.Seize(1)
        SimFunctions.SchedulePlus(Calendar,
            "EntryAgentEndOfService",
            SimRNG.Normal(RegMean,RegVar,2),Fax)
    else:
        EntryAgentsQueue.Add(Fax)

def EntryAgentEndOfService(DepartingFax):
    '''
    Called when a fax finishes service at an Entry Agent

    Randomly sends 20% of faxes to Specialist
    If no Specialist needed, fax leaves and its
        time-in-system statistics are recorded

    If an Entry Agent is free and queue nonempty, start service
        on next fax, otherwise, free an Entry Agent
    '''

    if SimRNG.Uniform(0,1,3) < 0.2:
        SpecialArrival(DepartingFax)
    else:
        TIS = SimClasses.Clock - DepartingFax.CreateTime
        RegTIS.Record(TIS)
        RegWithin10.Record((TIS < 10))

    if (EntryAgentsQueue.NumQueue() > 0 and
        EntryAgents.NumberOfUnits >= EntryAgents.CurrentNumBusy):
        NextFax = EntryAgentsQueue.Remove()
        SimFunctions.SchedulePlus(Calendar,
            "EntryAgentEndOfService",
            SimRNG.Normal(RegMean,RegVar,2),NextFax)
    else:
        EntryAgents.Free(1)
```

Fig. 4.28 Entry Agents functions

```
def SpecialArrival(SpecialFax):
    '''
    Called when a fax finishes service at an Entry Agent
        (specifically after EntryEntryAgentEndOfService function
        is called) and requires service by a Specialist

    If a Specialist is free, the fax seizes 1 unit
        and starts service, otherwise, the fax is added to
        SpecialistsQueue
    '''

    if Specialists.CurrentNumBusy < Specialists.NumberOfUnits:
        Specialists.Seize(1)
        SimFunctions.SchedulePlus(Calendar,
            "SpecialistEndOfService",
            SimRNG.Normal(SpMean,SpVar,4),SpecialFax)
    else:
        SpecialistsQueue.Add(SpecialFax)

def SpecialistEndOfService(DepartingFax):
    '''
    Called when a fax finishes service at a Specialist

    Fax leaves and its time-in-system statistics are recorded

    If a Specialist is free and queue nonempty, start service
        on next fax, otherwise, free a Specialist
    '''

    TIS = SimClasses.Clock - DepartingFax.CreateTime
    SpTIS.Record(TIS)
    SpWithin10.Record((TIS < 10))

    if (SpecialistsQueue.NumQueue() > 0 and
        Specialists.NumberOfUnits >= Specialists.CurrentNumBusy):
        NextFax = SpecialistsQueue.Remove()
        SimFunctions.SchedulePlus(Calendar,
            "SpecialistEndOfService",
            SimRNG.Normal(SpMean,SpVar,4),NextFax)
    else:
        Specialists.Free(1)
```

Fig. 4.29 Specialists functions

statistics. The function `SpecialArrival` corresponds to internal arrivals in the fax center queueing system, because "special" arrivals come from finishing service at the `EntryAgents` resource.

Next we discuss the function `ChangeStaff`, which is called at a change staff type event occurring at noon (240 min into the simulation run). Here we use the `SetUnits` method of the `Resource` to change the staffing levels of both the `EntryAgents` and `Specialists` resources. As an exercise, we leave it to

```
def ChangeStaff():
    '''
    Called at "ChangeStaff" event at noon

    Updates units for EntryAgents and Specialists
    If the number of EntryAgents increases after
        staff change, new EntryAgents begin
        service on faxes in EntryAgentsQueue,
        if any
    Similarly for Specialists
    '''

    EntryAgents.SetUnits(NumEntryAgentsPM)
    Specialists.SetUnits(NumSpecialistsPM)

    while (EntryAgentsQueue.NumQueue() > 0
        and (EntryAgents.NumberOfUnits
            >= EntryAgents.CurrentNumBusy)):

        NextFax = EntryAgentsQueue.Remove()
        EntryAgents.Seize(1)

        SimFunctions.SchedulePlus(Calendar,
            "EntryAgentEndOfService",
            SimRNG.Normal(RegMean,RegVar,2),NextFax)

    while (SpecialistsQueue.NumQueue() > 0
        and (Specialists.NumberOfUnits
        >= Specialists.CurrentNumBusy)):

        NextFax = SpecialistsQueue.Remove()
        Specialists.Seize(1)

        SimFunctions.SchedulePlus(Calendar,
            "SpecialistEndOfService",
            SimRNG.Normal(SpMean,SpVar,4),NextFax)
```

Fig. 4.30 Change staff function

the reader to verify that the function `ChangeStaff` successfully executes its intention and in particular allows any Entry Agent or Specialist working on a fax at noon to finish their service. We recommend considering the two types of staff changes: if `NumEntryAgentsAM` is less than `NumEntryAgentsPM`, or if `NumEntryAgentsAM` is more than `NumEntryAgentsPM`. The staff change for the number of Specialists obeys analogous logic. As a hint for the case in which `NumEntryAgentsAM` is more than `NumEntryAgentsPM`, the “if” statement in `EntryAgentEndOfService` is important (see Fig. 4.28).

```
for reps in range(0,NumReps):

    # Initialization
    # Reset number of EntryAgents and Specialists to AM staffing
    # Schedule first arrival and ChangeStaff event at noon
    SimFunctions.SimFunctionsInit(Calendar)
    EntryAgents.SetUnits(NumEntryAgentsAM)
    Specialists.SetUnits(NumSpecialistsAM)
    SimFunctions.Schedule(Calendar, "Arrival", NSPP(1))
    SimFunctions.Schedule(Calendar, "ChangeStaff",
        (NPeriods/2)*PeriodLength)

    # Main simulation loop
    while Calendar.N() > 0:
        NextEvent = Calendar.Remove()
        SimClasses.Clock = NextEvent.EventTime

        if NextEvent.EventType == "Arrival":
            Arrival()
        elif NextEvent.EventType == "EntryAgentEndOfService":
            EntryAgentEndOfService(NextEvent.WhichObject)
        elif NextEvent.EventType == "SpecialistEndOfService":
            SpecialistEndOfService(NextEvent.WhichObject)
        elif NextEvent.EventType == "ChangeStaff":
            ChangeStaff()

    # At end of replication, store stats in lists
    RegTISAvg.append(RegTIS.Mean())
    EntryAgentsQueueAvg.append(EntryAgentsQueue.Mean())
    RegPropWithin10.append(RegWithin10.Mean())

    SpTimeAvg.append(SpTIS.Mean())
    SpecialistsQueueAvg.append(SpecialistsQueue.Mean())
    SpPropWithin10.append(SpWithin10.Mean())

    EntryAgentsBusyAvg.append(EntryAgents.Mean())
    SpecialistsBusyAvg.append(Specialists.Mean())
    EndingTime.append(SimClasses.Clock)
```

Fig. 4.31 Main program for service center simulation

The main program for the simulation is in Fig. 4.31, which ties everything together. In each replication, we initialize the `EntryAgents` and `Specialists` resource units to their AM staffing levels and put two events on the event calendar: the first arrival and the change staff event at noon. While the event calendar is nonempty, we move through the event calendar events, and each event is either an arrival, Entry Agent end-of-service, Specialist end-of-service, or a change staff event. We leave it to the reader to verify that terminating the simulation replication when the event calendar is empty satisfies the desired properties in the problem

```
output = pandas.DataFrame(
    {"RegTISAvg": RegTISAvg,
    "EntryAgentsQueueAvg": EntryAgentsQueueAvg,
    "EntryAgentsBusyAvg":EntryAgentsBusyAvg,
    "SpTimeAvg": SpTimeAvg,
    "SpecialistsQueueAvg": SpecialistsQueueAvg,
    "SpecialistsBusyAvg": SpecialistsBusyAvg,
    "RegPropWithin10": RegPropWithin10,
    "SpPropWithin10": SpPropWithin10,
    "EndingTime": EndingTime})
output.to_csv("output.csv", sep=",")

print("Means")
print(output.mean())

print("95% CI Half-Width")
print(1.96*numpy.sqrt(output.var(ddof = 0)/len(output)))
```

Fig. 4.32 Main program for service center simulation

statement in Sect. 4.6.1. As usual we provide a snippet in Fig. 4.32 with a simple script for output generation.

Ten replications of this simulation with a staffing policy of 15 Entry Agents in the morning and 9 in the afternoon and 6 Specialists in the morning and 3 in the afternoon give 0.98 ± 0.02 for the fraction of regular faxes entered in 10 min or less, and 0.79 ± 0.08 for the special faxes. The "$\pm$" are 95% confidence interval halfwidths. Our stated policy appears to be close to the requirements, although if we absolutely insist on 80% of special faxes finishing in 10 min, then additional replications are needed to narrow the confidence interval.

Exercises

1. Simulate an $M(t)/G/\infty$ queue, where G corresponds to an Erlang distribution with fixed mean but try different numbers of phases. That is, keep the mean service time fixed but change the variability. Is the expected number in queue sensitive to the variance in the service time?
2. This problem assumes a more advanced background in stochastic processes. In the simulation of the $M(t)/M/\infty$ queue, there could be a very large number of events on the event calendar: one "Arrival" and one "Departure" for *each* car currently in the garage. However, properties of the exponential distribution can reduce this to no more than two events. Let $\beta = 1/\tau$ be the departure rate for a car (recall that τ is the mean parking time). If at any time we observe that there are N cars in the garage (no matter how long they have been there), then the time until the first of these cars departs is exponentially distributed

with mean $1/(N\beta)$. Use this insight to build an $M(t)/M/\infty$ simulation with at most two pending events, next arrival and next departure. *Hint*: Whenever an arrival occurs the distribution of the time until the next departure changes, so the scheduled next departure time must again be generated.

3. For the hospital problem, simulate the current system in which the receptionist's service time is well modeled as having an Erlang-4 distribution with mean 0.6 min. Compare the waiting time to the proposed electronic kiosk alternative.
4. Beginning with the event-based $M/G/1$ simulation, implement the changes necessary to make it an $M/G/s$ simulation (a single queue with any number of servers). Keeping $\lambda = 1$ and $\tau/s = 0.8$, simulate $s = 1,2,3$ servers and compare the results. What you are doing is comparing queues with the same service capacity, but with 1 fast server as compared to two or more slower servers. State clearly what you observe.
5. Modify the PythonSim event-based simulation of the $M/G/1$ queue to simulate an M/G/1/c retrial queue. This means that customers who arrive to find c customers in the system (including the customer in service) leave immediately but arrive again after an exponentially distributed amount of time with mean `MeanTR`. *Hint*: The existence of retrial customers should not affect the arrival process for first-time arrivals.
6. Modify the SAN simulation to allow each activity to have a different mean time to complete (currently they all have mean time 1). Use a collection to hold these mean times.
7. Try the following numbers of steps for approximating the value of the Asian option to see how sensitive the value is to the step size: $m = 8,16,32,64,128$.
8. In the simulation of the Asian option, the sample mean of 10,000 replications was 2.198270479, and the standard deviation was 4.770393202. Approximately how many replications would it take to decrease the relative error to less than 1%?
9. For the service center, increase the number of replications until you can be confident that the suggested policy does or does not achieve the 80% entry in less than 10 min requirement for special faxes.
10. For the service center, find the minimum staffing policy (in terms of the total number of staff) that achieves the service-level requirement. Examine the other statistics generated by the simulation to make sure you are satisfied with this policy.
11. For the service center, suppose that Specialists earn twice as much as Entry Agents. Find the minimum cost staffing policy that achieves the service-level requirement. Examine the other statistics generated by the simulation to make sure you are satisfied with this policy.
12. For the service center, suppose that the staffing level can change hourly, but once an Agent or Specialist comes on duty they must work for 4 h. Find the minimum staffing policy (in terms of the total number of staff) that achieves the service-level requirement.
13. For the service center, pick a staffing policy that fails to achieve the service-level requirements by 20% or more. Rerun the simulation with a replication

being defined as exactly 8 h, but do not carry waiting faxes over to the next day. How much do the statistics differ using the two different ways to end a replication?

14. The function `NSPP` is presented in Fig. 4.27. This function implements the thinning method described in Sect. 4.2 for a nonstationary Poisson process with piecewise-constant rate function. Study it and describe how it works.
15. The phone desk for a small office is staffed from 8 a.m. to 4 p.m. by a single operator. Calls arrive according to a Poisson process with rate 6 per hour, and the time to serve a call is uniformly distributed between 5 and 12 min. Callers who find the operator busy are placed on hold, if there is space available; otherwise, they receive a busy signal and the call is considered "lost." In addition, 10% of callers who do not immediately get the operator decide to hang up rather than go on hold; they are not considered lost, since it was their choice. Because the hold queue occupies resources, the company would like to know the smallest capacity (the number of callers) for the hold queue that keeps the daily fraction of lost calls under 5%. In addition, they would like to know the long-run utilization of the operator to make sure he or she will not be too busy. Use PythonSim to simulate this system and find the required capacity for the hold queue. Model the callers as class `Entity`, the hold queue as class `FIFOQueue` and the operator as class `Resource`. Use the PythonSim functions `Expon` and `Uniform` for random-variate generation. Use class `DTStat` to estimate the fraction of calls lost (record a 0 for calls not lost and a 1 for those that are lost so that the sample mean is the fraction lost). Use the statistics collected by class `Resource` to estimate the utilization.
16. Software Made Personal (SMP) customizes software products in two areas: financial tracking and contact management. They currently have a customer support call center that handles technical questions for owners of their software from the hours of 8 a.m. to 4 p.m. Eastern time.

    When a customer calls, they first listen to a recording that asks them to select among the product lines; historically, 59% are financial products and 41% contact management products. The number of customers who can be connected (talking to an agent or on hold) at any one time is essentially unlimited. Each product line has its own agents. If an appropriate agent is available, then the call is immediately routed to the agent, and if an appropriate agent is not available, then the caller is placed in a hold queue (and listens to a combination of music and ads). SMP has observed that hang-ups very rarely happen.

    SMP is hoping to reduce the total number of agents they need by cross-training agents so that they can answer calls for any product line. Since the agents will not be experts across all products, this is expected to increase the time to process a call by about 5%. The question that SMP has asked you to answer is how many cross-trained agents are needed to provide service at the same level as the current system.

    Incoming calls can be modeled as a Poisson arrival process with a rate of 60 per hour. The mean time required for an agent to answer a question is 5 min, with the actual time being Erlang-2 for financial calls and Erlang-3 for contact management calls. The current assignment of agents is four for financial and three for contact management. Simulate the system to find out how many agents are needed to deliver the same level of service in the cross-trained system as in the current system.

17. Beginning with the `FIFOQueue` class module, develop a new class module called `PriorityQueue`. When an `Entity` is added to the `Priority-Queue`, its position depends on the entity's `Priority` property with high value of `Priority` first. Be sure to add the `Priority` property to the `Entity` class module.
18. The fax entry times were modeled as being normally distributed. However, the normal distribution admits negative values, which certainly does not make sense. What should be done about this? Consider mapping negative values to 0 or generating a new value whenever a negative value occurs. Which is more likely to be realistic and why?

Chapter 5
Three Views of Simulation

Prior to this chapter, the focus of the book has been on building—literally writing computer code for—simulation models. This chapter sets up the remainder of the book which addresses issues related to experiment design and analysis. To do so, we present three different, but complementary ways of viewing computer simulation. Section 5.1 makes clear the roles of real systems, simulated systems, and the conceptual systems that the analyst wants to design. This framework highlights sources of error in a simulation study; it is abstract, but not mathematically formal. Section 5.2 views simulation output as a stochastic process to provide a framework for designing simulation experiments and analyzing the results; it is a mathematical treatment that provides the foundation for statistical analysis. Section 5.3 describes simulation from a computational perspective with an emphasis on how computation impacts experiment design and analysis. All of these perspectives are necessary to have a deep understanding of stochastic simulation.

5.1 A Framework for Simulation Modeling and Analysis

Example 5.1 (The Call Center). A software company has a customer call center for inquiries or problems related to its products. Currently the agents are trained only for particular products, and therefore they only handle calls related to their product. The company would like to reduce staff, while still delivering the same level of service, by cross-training some agents to handle calls regarding more than one product. What staff level is required?

This situation has characteristics that are common to many system-design problems for which we seek a simulation solution: There is a conceptual design for a new system (the call center with cross-trained agents). There is also an existing real system that is related to the conceptual one, but is not identical to it (the current

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_5

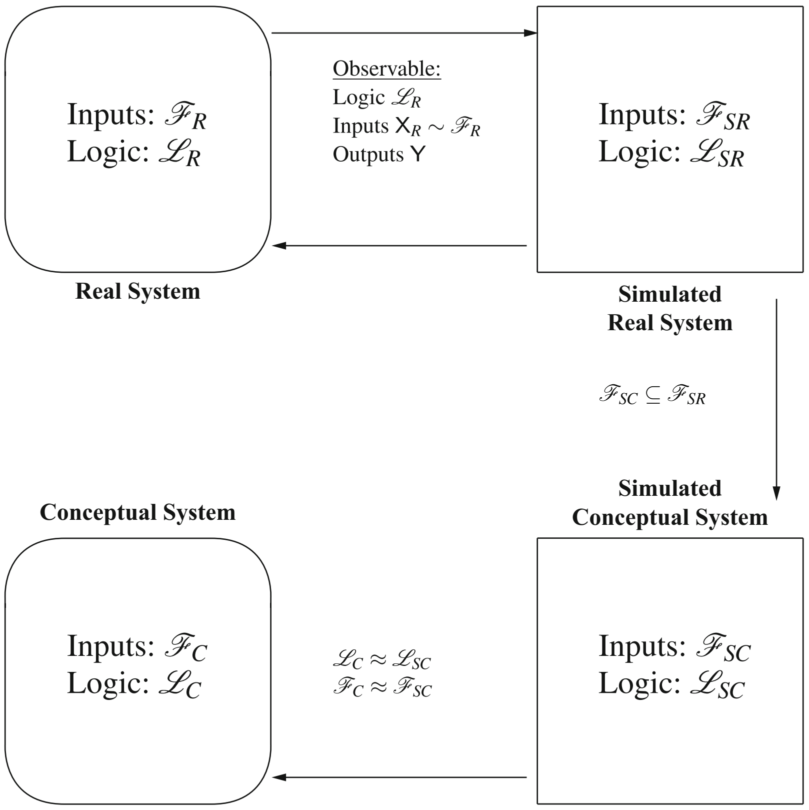
Inputs: $\mathscr{F}_R$
Logic: $\mathscr{L}_R$
Real System
Observable:
Logic $\mathscr{L}_R$
Inputs $\mathsf{X}_R \sim \mathscr{F}_R$
Outputs Y
Inputs: $\mathscr{F}_{SR}$
Logic: $\mathscr{L}_{SR}$
Simulated Real System
$\mathscr{F}_{SC} \subseteq \mathscr{F}_{SR}$
Simulated Conceptual System
Conceptual System
Inputs: $\mathscr{F}_C$
Logic: $\mathscr{L}_C$
$\mathscr{L}_C \approx \mathscr{L}_{SC}$
$\mathscr{F}_C \approx \mathscr{F}_{SC}$
Inputs: $\mathscr{F}_{SC}$
Logic: $\mathscr{L}_{SC}$

Fig. 5.1 Relationship between real, simulated, and conceptual systems

call center with dedicated agents). Using the real system as a starting point, a simulated system will be constructed to evaluate how well the conceptual system might work. In this section we describe a framework for simulation problems such as the call center, a framework that will be informative as we think about simulation input modeling, output analysis, and experiment design (the subjects of Chaps. 6–9).

For the purpose of this framework, a *system* consists of *inputs* and *logic*. Inputs are the uncertain (stochastic) components of a system, while the logic may be thought of as a collection of rules or algorithms that govern how the system behaves as a function of the inputs. Let $\mathscr{L}$ denote logic, and let $\mathscr{F}$ denote the probability model that governs the inputs (often a collection of probability distributions). In our framework, there are three different kinds of systems: Real systems (R), simulated systems (S), and conceptual systems (C); by a "conceptual" system we usually mean a system that could exist, but does not yet exist. Systems have performance parameters that are implied by the inputs and logic; we will denote these generically by μ. Refer to Fig. 5.1 for the discussion that follows.

An *experiment* on a system, denoted by $\mathscr{E}$, is a recipe for exercising the system to observe its behavior. Experiments on real systems will often be field observations of the system as it currently operates, because a designed experiment on a real system might be expensive, disruptive, or dangerous. Vast quantities of data are passively collected on many real systems automatically. Simulation experiments, on the other hand, should be designed using statistical principles because the simulated system is completely under our control. Experiments on a conceptual system are, by definition, impossible, which is why we do simulations.

An experiment on a system generates observed behavior, which for our purposes is always numerical. Denote numerical observations of a system's inputs by X and its outputs by Y, both collections of random variables in general. Outputs are derived quantities that are a function of the inputs, logic, and experiment design. It is very important to understand that Y is not well defined without an experiment, $\mathscr{E}$. Observing a real system for a month is different than observing it for a day; running a simulation for ten replications is different than running it for 1000. This is critical, because performance parameters μ are estimated by functions of the outputs, $\mathscr{T}(\mathsf{Y})$, and therefore properties of these estimators depend on the experiment design.

For example, in the current call center the inputs $\mathscr{F}_R$ include the arrival process of calls, and the time required to resolve a caller's problem (excluding the time they spend on hold before talking to an agent). These are inputs because they are uncertain, stochastic quantities that we do not view as being functions of some more basic stochastic quantities. The call center logic $\mathscr{L}_R$ includes the hours of operation and the rules for how calls are routed to agents. A performance parameter μ_R that measures customer satisfaction is the long-run average time a caller spends on hold before talking to an agent. Therefore, an output of interest Y_R is how long individual callers spend on hold. And an experiment $\mathscr{E}_R$ might be logging data on call center activity (call arrival times, times to resolve problems, and caller hold times) for several months.

In our typical situation, there is a conceptual system $\{\mathscr{L}_C, \mathscr{F}_C\}$ whose performance μ_C we would like to predict; in the example it is the call center with cross-trained agents that we might want to implement. If we could perform an experiment on this system, then we could use observed outputs Y_C to estimate μ_C (e.g., the mean hold time when using cross-trained agents). Of course, we cannot do this directly, because the system is conceptual. We can, however, perform an experiment on the real system $\{\mathscr{L}_R, \mathscr{F}_R\}$; specifically, we can study its logic $\mathscr{L}_R$, and observe its inputs X_R and outputs Y_R. We do this to aid in developing a simulation of the conceptual system $\{\mathscr{L}_{SC}, \mathscr{F}_{SC}\}$. Notice that while the rules that govern a system's behavior $\mathscr{L}_R$ may be at least partially observable, the input probability model $\mathscr{F}_R$ is not; only the input data itself X_R can be observed. For instance, we can log the times that calls arrive, but not the probability mechanism that generated the calls.

Simulation is useful for evaluating a conceptual system design when we believe we understand the critical aspects of the conceptual system's logic $\mathscr{L}_C$ (often that logic is our choice, like using cross-trained agents), and we also believe that $\mathscr{F}_C \subseteq \mathscr{F}_R$—that is, the input processes for the conceptual system are a subset of the input processes for the real system. For instance, using cross-trained agents likely will have no impact on the arrival process of calls. This leads naturally to constructing two simulated systems:

- $\mathsf{SR} = \{\mathscr{L}_{SR}, \mathscr{F}_{SR}\}$: This is the simulation of the real system (SR stands for "simulated-real" system). Ideally this simulated system has performance parameters μ_{SR} that are close to those of the real system μ_R so we can validate it.
- $\mathsf{SC} = \{\mathscr{L}_{SC}, \mathscr{F}_{SC}\}$: This is the simulation of the conceptual system (SC stands for "simulated-conceptual" system). Ideally this simulated system has performance parameters μ_{SC} that are close to those of the conceptual system μ_C. Recall that the input models for the simulated-conceptual system are a subset of the input models for the simulated-real system, while the conceptual system's logic is our choice.

With this framework, we can now define the simulation design and analysis problems that will occupy the remainder of the book:

Input modeling: This is the process of choosing the simulation input probability models $\mathscr{F}_{SR}$ that approximate the input models of the real system $\mathscr{F}_R$. When real system inputs X_R can be observed, then we may fit distributions to the data or simply resample the data; otherwise we have to use subjective methods. Similarly, input models for the simulated conceptual system $\mathscr{F}_{SC}$ must be chosen.

Variate generation: Given the simulation input models $\mathscr{F}_{SR}$ and $\mathscr{F}_{SC}$, we need methods to generate realizations from them to drive the simulation. We touched on this topic in Sect. 2.2, but will go into more depth and describe variate-generation algorithms for more complex input models in the following chapter.

Validation: In theory, this is the process of deciding whether the performance parameters of the simulated-conceptual system μ_{SC} are close enough to the performance parameters of the conceptual system μ_C for the simulation to be useful for the decisions that it needs to support. Unfortunately, this sort of validation is impossible to achieve by any quantitative means. For this reason we often construct a simulation of the real system, $\{\mathscr{L}_{SR}, \mathscr{F}_{SR}\}$, because we can validate it with respect to the real system $\{\mathscr{L}_R, \mathscr{F}_R\}$. Knowing we can model the real system gives us confidence that our simulation of the conceptual system will also be useful (although this is certainly not a guarantee).

Our definition of validation as a quantitative evaluation of $\{\mathscr{L}_{SR}, \mathscr{F}_{SR}\}$ with respect to $\{\mathscr{L}_R, \mathscr{F}_R\}$ is narrower than many researchers who also provide non-quantitative ways to validate $\{\mathscr{L}_{SC}, \mathscr{F}_{SC}\}$ with respect to $\{\mathscr{L}_C, \mathscr{F}_C\}$. For a broader perspective, see Sargent (2011) and Robinson (2014).

Output analysis: The goal of the simulation experiment is to estimate performance parameters μ_{SC} (which we hope are close to those of the conceptual system μ_C). Output analysis is concerned with designing a simulation experiment $\mathscr{E}_{SC}$ for $\{\mathscr{L}_{SC}, \mathscr{F}_{SC}\}$, and estimating the performance characteristics μ_{SC} via the generated outputs Y_{SC}. If, as is often the case, there is more than one conceptual system, then output analysis may include designing an experiment to identify the best conceptual system.

With this framework in place, the sources of error in solving a problem via simulation can be defined.

Modeling error: $\mathscr{L}_{SC} \neq \mathscr{L}_C$ or $\mathscr{L}_{SR} \neq \mathscr{L}_R$. There is almost always modeling error, since the real or conceptual system will typically be richer and more complicated than the simulation model, with behaviors that cannot be quantified by a collection of algorithmic rules. For instance, in the call center example the rule for assigning incoming calls to agents may be well defined, while the rule for passing a call along to a supervisor is more ambiguous and agent dependent. The fiction that a real system has quantifiable logic is necessary for modeling, but is not strictly true. The key is whether the logic we use in our simulation model is accurate enough to obtain useful results. Simulation logic was the focus of Chaps. 1–4.

Input uncertainty: $\mathscr{F}_{SC} \neq \mathscr{F}_C$ or $\mathscr{F}_{SR} \neq \mathscr{F}_R$. The true input distributions $\mathscr{F}_R$ are only discernible through the observed input data X_R, which is always a finite sample. We typically fit $\mathscr{F}_{SR}$ to these data, so there will be error. The problem is compounded when we move to $\mathscr{F}_{SC}$. Again, the existence of "true" distributions $\mathscr{F}_C$ and $\mathscr{F}_R$ is not strictly correct, but is the basis of all of statistics, and it allows input modeling to be treated as a statistical problem.

Estimation error: $\mathscr{T}(\mathsf{Y}_{SC}) \neq \mu_{SC}$. Experiments on simulated systems are statistical experiments because they generate realizations of the output random variables Y_{SC} to estimate performance properties μ_{SC}; but again, only with a finite sample. Therefore, we need to account for estimation error, often via confidence intervals for the unknown performance parameters. Estimation error is the easiest to quantify, but there can still be complications due to bias and correlated data.

"Uncertainty quantification," "model risk," "sensitivity analysis," and "calibration" are broad terms that address various aspects of the errors cited above, including ways to measure them, hedge against them or adjust for them. This is an important research area with significant practical consequences. General references include the *SIAM/ASA Journal on Uncertainty Quantification* and textbooks such as Gramacy (2020) and Saltelli et al. (2008).

Clearly, not all simulation studies have all the features of the framework described here. For instance, input modeling without any real data is a common occurrence. Also, there may be no real system, only one or more conceptual ones that we want to evaluate. These are special cases of our general framework.

5.2 Simulation as a Stochastic Process

In this section we focus on simulation output processes, the data generated by a simulation that are used to estimate system performance. We think of simulation output as a *stochastic process*: an indexed sequence of random variables defined on a common probability space. Depending on the output process, the index may be time, observation count or replication number. We are particularly interested in large-sample behavior of summary statistics as the index (time, observation count, or number of replications) increases. Large-sample, or asymptotic, behavior is relevant for simulation as we are frequently able to simulate very large samples, unlike, for instance, expensive physical experiments. To set up the formal description of asymptotic behavior, we start with a less formal simulation description.

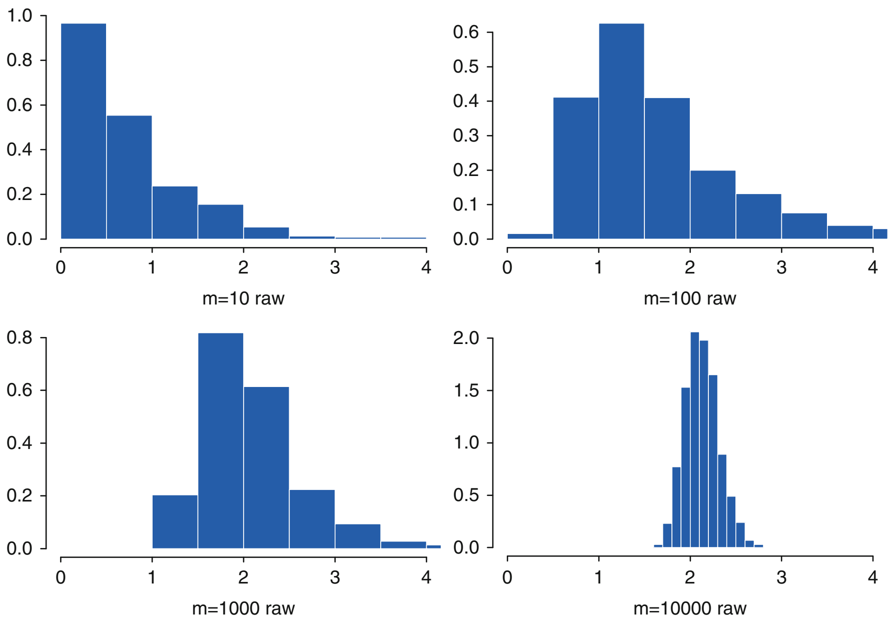

Fig. 5.2 Histogram of the 1000 replication averages at $m = 10, 100, 1000$, and 10,000

5.2.1 *Simulated Asymptotic Behavior*

To illustrate the key aspects of asymptotic behavior we will use the $M/G/1$ queue example of Sects. 3.2 and 4.3; recall that this is a single-server queue that was used to model a hospital reception system to estimate long-run average delay in queue, which we will denote by μ. In the simulation we observed $Y_1, Y_2, \ldots, Y_m$, the delay in queue of the first m patients and visitors, and estimated the long-run average by the sample mean $\bar{Y}(m) = \sum_{i=1}^{m} Y_i/m$. We will consider the same case as in Sect. 4.3 where the arrival rate is one customer per minute, and the service time has an Erlang distribution with mean $\tau = 0.8$ min and three phases. For this model the Pollaczek–Khinchine formula in Eq. (3.4) gives $\mu = 2.133$ min, so of course simulation is not actually needed.

Now suppose that you and 999 of your friends are hired by "the boss" to each simulate one replication of this system for $m = 10{,}000$ patients and visitors, choosing your initial random-number seeds independently, randomly, and without collaboration. However, because your boss is eager to know the outcome, you will each report your results to her at check points $m = 10, 100, 1000$ and finally 10,000 simulated patients and visitors. That is, each of you will report $\bar{Y}(10)$, $\bar{Y}(100)$, $\bar{Y}(1000)$ and $\bar{Y}(10{,}000)$. *You* only get to see the result of your one replication, but your boss gets to see them all. We will look at the results both from your and your boss's perspectives.

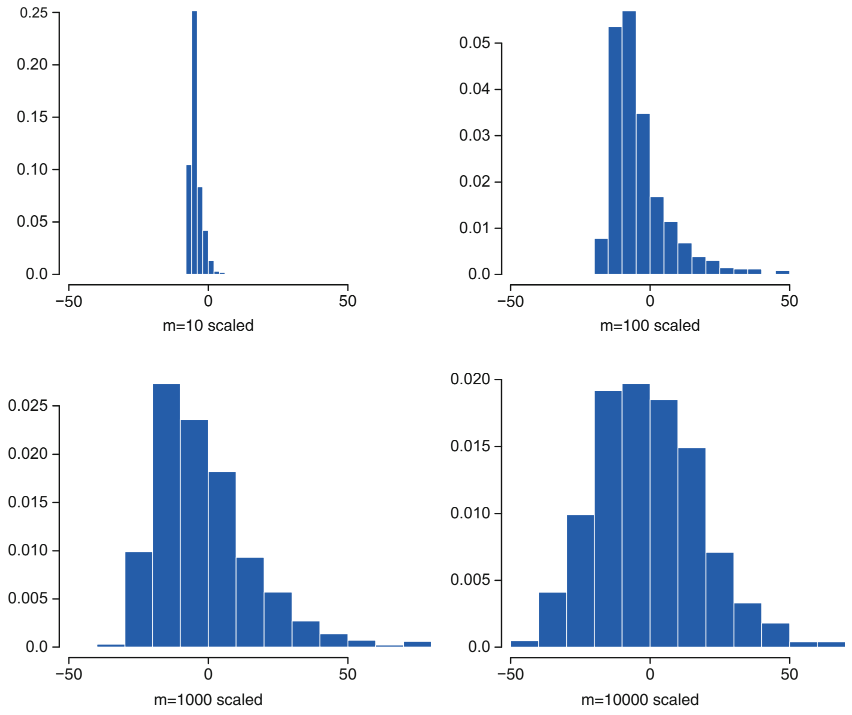

Fig. 5.3 Histogram of the 1000 scaled and centered replication averages at $m = 10, 100, 1000$, and 10,000

Figure 5.2 shows the histograms that your boss would see of the 1000 $\bar{Y}(10)$, $\bar{Y}(100)$, $\bar{Y}(1000)$ and $\bar{Y}(10{,}000)$ values. Your boss can see that the variability in the results you and your friends submit decreases as m increases. That is, they cluster more closely around a central value. Also, the shape of the histogram becomes more nearly symmetric and bell-shaped although it appears to be decreasing to a point.

If your boss instead plotted a histogram of the 1000 $\sqrt{m}\,(\bar{Y}(m) - 2.133)$ values for each m—that is, she took each sample mean $\bar{Y}(m)$, subtracted the true steady-state mean $\mu = 2.133$, and multiplied the result by $\sqrt{m}$—then she would see the sequence of histograms in Fig. 5.3. Notice that rather than the variability of the histograms decreasing, as happened with the raw averages, the distribution appears to be stabilizing toward a specific mean-zero (perhaps) normal distribution. This is an example of convergence in distribution as promised by the central limit theorem, which we state precisely below. We often say loosely that "the sample means become normally distributed," but as these figures show, the scaling up by $\sqrt{m}$ is needed to converge to a stable distribution (one whose variability is not shrinking as in Fig. 5.2).

Table 5.1 Summary measures of the 1000 sample means

m	10	100	1,000	10,000
$\bar{\bar{Y}}(m)$	0.718	1.739	2.083	2.130
$\bar{\bar{Y}}(m) - 2.133$	−1.415	−0.394	−0.0503	−0.003
$\dfrac{\#\{\lvert\bar{Y}(m) - 2.133\rvert > 0.5\}}{1,000}$	0.914	0.742	0.359	0.010

Still looking at things from your boss's point of view, we will summarize the sample averages in three additional ways: $\bar{\bar{Y}}(m)$, the overall average of the 1000 averages obtained at each check point; $\bar{\bar{Y}}(m) - 2.133$, the difference between the overall average and the true steady-state mean; and the fraction of the 1000 averages that differ from $\mu = 2.133$ min by more than 0.5 min. These three measures are empirical estimates of $\mathrm{E}(\bar{Y}(m))$, $\mathrm{Bias}(\bar{Y}(m))$ and $\Pr\{|\bar{Y}(m) - 2.133| > 0.5\}$. The results are shown in Table 5.1. Notice that as m increases, the overall mean seems to be converging to 2.133, the bias seems to be going to 0, and the likelihood of a sample mean being more than 0.5 min from 2.133 seems to be going to zero. Notice also that the bias is negative, a result of starting the simulation with no patients and visitors in the system so that for small m the delay in queue tends to be quite low.

Your sample means contributed to these results; they were 0.517, 2.212, 1.621, and 1.972, for $m = 10, 100, 1000$ and 10,000, respectively, each just one of the 1000 sample means your boss observed at each check point. None of them equal 2.133, meaning they all have error, consisting of bias and sampling variability. What does large-sample asymptotic behavior tell you about *your* single result? Knowing how things look from your boss' point of view helps provide an interpretation of the various modes of convergence as they apply to you; we define these now and relate them back to the example.

Consider a sequence of random variables $\{W_1, W_2, \ldots\}$ and a constant μ. To relate the situation to the queueing problem, let $W_m = \bar{Y}(m), m = 1, 2, \ldots$, be *your* sequence of sample means.

Definition 5.1. Convergence in probability: $W_m \xrightarrow{P} \mu$ if for any $\varepsilon > 0$,

$$\lim_{m\to\infty} \Pr\{|W_m - \mu| > \varepsilon\} = 0.$$

In the queueing example we noticed that as m increases, fewer and fewer of the sample means your boss saw were greater than $\varepsilon = 0.5$ away from $\mu = 2.133$. While you could be unlucky with your one result, convergence in probability says that the chances that you are unlucky become smaller and smaller. Your last sample mean, $\bar{Y}(10{,}000) = 1.972$, is less than 0.5 from μ.

Definition 5.2. Convergence with probability 1: $W_m \xrightarrow{a.s.} \mu$ if

$$\Pr\left\{\lim_{m\to\infty} W_m = \mu\right\} = 1.$$

The abbreviation a.s. stands for "almost surely," so convergence with probability 1 is also called almost sure convergence.

To really make the definition of a.s. convergence complete we also need to define the underlying probability space on which the sequence W_m is defined. However, a simulation analogy captures the essence of this type of convergence: Recall that you and your friends independently and randomly selected a random-number seed for your queueing simulation. Once you pick a seed your simulation outputs are a deterministic sequence of numbers. Almost sure convergence means that with probability 1 you will select a seed such that your sequence of sample means $\bar{Y}(m)$ converges to μ in the usual sense of convergence of a sequence of numbers to a constant. Some of your friends may get there faster, and others slower, but you are all guaranteed that as m increases *your* sample mean $\bar{Y}(m)$ will converge to μ.[1]

Although we have defined convergence in probability and with probability 1 as convergence of a sequence of random variables to a constant, they can also be extended to convergence to a random variable W defined on the same probability space as W_m. However, we do not need the more general definition just yet.

To describe convergence in distribution, and specifically the central limit theorem, consider a sequence of random variables $Z_1, Z_2, \ldots$ and a random variable Z whose distribution is not a function of m. Let $F_m(z) = \Pr\{Z_m \le z\}$ and let $F(z) = \Pr\{Z \le z\}$. In the queueing simulation we could let $Z_m = \sqrt{m}(\bar{Y}(m) - \mu)$.

Definition 5.3. Convergence in distribution: $Z_m \xrightarrow{D} Z$ if

$$\lim_{m\to\infty} F_m(z) = F(z)$$

for all points z at which $F(z)$ is continuous. This definition extends without change to Z_m and Z being vector-valued random variables.

In the queueing simulation, the boss saw that as m got large the empirical distribution of the scaled and centered sample means became a stable, mean-zero normal distribution. Thus, F is a mean-zero normal distribution. When convergence in distribution is established to a mean-zero normal distribution it is often referred to as a "central limit theorem."

Central limit theorems are extremely valuable. If one holds for $Z_m = \sqrt{m}(\bar{Y}(m) - \mu)$, then for large m we can say that $\sqrt{m}(\bar{Y}(m) - \mu) \approx \gamma\mathrm{N}(0,1)$, where $\mathrm{N}(0,1)$ is a standard normal random variable and γ^2 is a (usually unknown) variance constant. Or equivalently, for large m

$$\bar{Y}(m) \approx \mu + \frac{\gamma}{\sqrt{m}}\mathrm{N}(0,1). \tag{5.1}$$

Thus, up to the variance constant γ^2 (which can often be estimated), the central limit theorem completely characterizes the error in using $\bar{Y}(m)$ to estimate μ. Specifically, it says that the error is symmetric about μ, and normally distributed, with standard deviation $\gamma/\sqrt{m}$. This characterization permits the derivation of bounds on the error that are correct with high probability (a confidence interval). The only

[1] While this is a good analogy, it is not precisely correct since random-number generators imitate truly random numbers and in fact repeat the same (long) cycle of values over and over again.

missing piece is the variance constant, which is why estimation of γ^2 has been such an important topic in simulation research.

5.2.2 Asymptotics Across Replications

The large-sample properties defined in the previous section are useful because they provide some assurance that our simulation-based estimates are converging to the performance measures of interest, and by helping to quantify the remaining error when we stop simulating. This leads to the obvious question: When can we expect such convergence properties to hold?

The most easily verifiable collection of theorems are for the i.i.d. case; in the queueing example, this is from your boss's point of view. Remember that your boss sees $n = 1000$ (or more if you find more friends) results, each independently simulated according to the same rules. In other words, your boss sees independent and identically distributed replications.

Theorem 5.1. Weak law of large numbers (WLLN): *If* $Z_1, Z_2, \ldots$ *are i.i.d. with* $\mathrm{E}(Z_1) < \infty$*, then*

$$\frac{1}{n}\sum_{i=1}^{n} Z_i \xrightarrow{P} \mathrm{E}(Z_1).$$

Strong law of large numbers (SLLN): *If* $Z_1, Z_2, \ldots$ *are i.i.d. with* $\mathrm{E}(Z_1) < \infty$*, then*

$$\frac{1}{n}\sum_{i=1}^{n} Z_i \xrightarrow{a.s.} \mathrm{E}(Z_1).$$

Central limit theorem (CLT): *If* $Z_1, Z_2, \ldots$ *are i.i.d. with* $\mathrm{E}(Z_1^2) < \infty$*, then*

$$\sqrt{n}\left(\frac{1}{n}\sum_{i=1}^{n} Z_i - \mathrm{E}(Z_1)\right) \xrightarrow{D} \gamma \mathrm{N}(0,1),$$

where $\gamma^2 = \mathrm{Var}(Z_1)$.

These results say that if your boss hired more and more friends to submit results, the average of these results would converge (both in probability and almost surely) to the mathematical expected value, and that the distribution of the (scaled) average would look more and more normally distributed.

There is a very subtle but important point to emphasize here: The centering constant in these theorems is $E(Z_1)$, which is not necessarily μ, the performance measure you are interested in estimating.[2] This is exactly the situation in the queueing example: For any finite m, the $E(Y(m)) \neq \mu$ (the steady-state mean) due to bias induced by the way the simulation was initialized empty and idle. As we saw

[2] Since $Z_1, Z_2, \ldots$ are i.i.d., we can select Z_1 or any of the others as the representative.

in Table 5.1, results are biased low (underestimating the steady-state mean waiting time). Since we will often make many replications to control statistical error, we need to remember that replications do not affect this sort of bias. In the next section we see how this sort of bias can be tamed by increasing the length of the replication (number of customers m in the queueing example).

What is "Z_1" in each of our other standard examples, and is $\mathrm{E}(Z_1)$ what we want to estimate?

Stochastic activity network: Here the goal was to estimate $\theta = \Pr\{Y > t_p\}$ and our observation on replication 1 is

$$Z_1 = \begin{cases} 1, & Y_1 > t_p \\ 0, & Y_1 \le t_p \end{cases}$$

Thus, $E(Z_1) = 1 \cdot \Pr\{Y_1 > t_p\} + 0 \cdot \Pr\{Y_1 \le t_p\} = \theta$; so the WLLN, SLLN, and CLT guarantee precisely the sort of convergence we want.

Asian option: Here the quantity of interest is

$$\nu = \mathrm{E}\left[\mathrm{e}^{-rT}\left(\bar{X}(T) - K\right)^+\right]$$

but what we observe is

$$Z_1 = \mathrm{e}^{-rT}\left(\widehat{\bar{X}_1(T)} - K\right)^+,$$

where $\widehat{\bar{X}_1(T)}$ is a discretized version of $\bar{X}(T)$ from replication 1; discretization introduces bias. Thus, $\mathrm{E}(Z_1) \ne \nu$, and no number of replications can cause $\sum_{i=1}^n Z_i/n$ to converge to ν; of course, it will converge to something not far from ν if the discretization error is small.

M(t)/M/∞ queue: Suppose that, as in Sect. 4.2, we are interested in the maximum number of cars in the parking lot over a 24-h period. Recall that $N(t)$ was the stochastic process representing the number of cars in the lot at time t. A quantity of interest then might be

$$\mu = \mathrm{E}\left(\max_{0\le t\le 24} N(t)\right)$$

the expected value of the maximum over a 24-h period. The natural output—as illustrated in Sect. 4.2—is

$$Z_1 = \max_{0\le t\le 24} N_1(t).$$

Therefore $\mu = \mathrm{E}(Z_1)$ by definition and the convergence results in this section imply that taking more replications guides us to the right value.

The WLLN, SLLN, and CLT show that sample means of i.i.d. data have very desirable properties under very general conditions. When we average across-replication outputs, these results apply. In our queueing example the replications came from you and your friends conducting independent simulations; more typically it is only you executing multiple replications with different random numbers. Assuming a good random-number generator, we treat these two ways of obtaining replications as the same. However, we have to take some care to understand the relationship between the expected value of our estimator and the expected value we are trying to estimate; when they are not the same, no quantity of replications can reduce the bias.

Remark 5.1. Convergence in probability and with probability 1 do not always hold together; if this was the case, then there would only be a need for one concept. The SLLN is called "strong" for a reason: it implies that the WLLN holds, but not the other way around. This important distinction is treated in advanced texts on probability such as Billingsley (1995).

The WLLN, SLLN and CLT establish convergence of the sample mean $\bar{Z}(n) = \sum_{i=1}^{n} Z_i/n$ to its expectation $\mu = \mathrm{E}(Z_1)$. **Large deviations (LD)** results provide deeper insight into how this happens. There are many LD results; we state two, one on the convergence to 0 of the probability of a large deviation, and the second on the probability of any large deviations of the entire sequence of sample means $\bar{Z}(n), n = 1, 2, \ldots$.

Theorem 5.2 (Cramér's Theorem). *Suppose that $Z_1, Z_2, \ldots$ are i.i.d. and Z has finite log moment generating function, that is $\ln[\mathrm{E}(e^{tZ})] < \infty$ for all t. Then for any $z > \mathrm{E}(Z)$ there exists a constant $I(z)$ that depends on the distribution of Z and the value z such that*

$$\lim_{n\to\infty} \frac{1}{n} \ln\left[\Pr\{\bar{Z}(n) \geq z\}\right] = -I(z). \tag{5.2}$$

This famous result implies that as n increases, the probability that $\bar{Z}(n)$ is above its mean decreases exponentially fast: $\Pr\{\bar{Z}(n) \geq z\} \approx e^{-nI(z)}$. As an example, if $Z \sim \mathrm{N}(\mu, \sigma^2)$, then $I(z) = (z-\mu)^2/(2\sigma^2)$. If one has a choice among unbiased estimators that satisfy Theorem 5.2, then the one with the larger rate constant $I(z)$ is preferred in terms of speed of convergence. A reference on LD methods with application to simulation is Bucklew (2004).

Theorem 5.3. *If $Z_1, Z_2, \ldots$ are i.i.d. normally distributed with mean $-\infty < \mu < 0$ and variance $\sigma^2 < \infty$, then for any constant $a > 0$ the*

$$\Pr\left\{\bar{Z}(n) > a/n \textit{ for some } n < \infty\right\} \leq e^{2\mu a/\sigma^2}. \tag{5.3}$$

The assumption that $\mu < 0$ implies that $\bar{Z}(n)$ tends to be negative. This result bounds the probability that it *ever* becomes positive by more than a specific amount that is tending to 0. Such results support sequential statistical procedures that examine the sequence $\bar{Z}(n)$ for many values of n. Notice that the assumption that Z is normally distributed is stronger than the assumptions of Theorem 5.2.

5.2.3 Asymptotics Within Replications

Recall that in the queueing experiment you only got to see one (quite long) replication. The results that your boss saw from across many replications suggested that you might be able to assume that a strong law of large numbers or central limit theorem applies to *your* particular $\bar{Y}(m)$ as the number m of patients and visitors, rather than the number n of replications, increases. However, your data are clearly not i.i.d. since the first patient waits 0 min no matter what, and Lindley's equation (3.3) shows that the waiting times of successive patients and visitors are dependent. Therefore, the theorems of Sect. 5.2.2 do not apply.

Unfortunately, there are no general, easily verifiable conditions for establishing convergence in the non-i.i.d. case. In fact, a rather deep understanding of the stochastic process being simulated is essential to actually prove a SLLN/WLLN or CLT for a dependent process. See Henderson (2006) and the references therein for a lucid, but technically challenging exploration of this point.

That said, the desired large-sample properties often hold. Typically heuristic justifications are offered based on analogies with simpler processes that have been studied: Stationary Markovian queues where the service capacity exceeds the customer load; stationary, irreducible, positively recurrent Markov chains; time series processes such as the AR(1); and regenerative processes are a few examples. The appendix to this chapter discusses regenerative structure in more detail; interested readers can refer to Haas (2002) and Glynn (2006).

Roughly, conditions such as the following must hold:

- Neither the logic of the system nor the input processes that drive the simulation can change over time, and there can be no periodic (cyclic) behavior.
- The state space of the simulation may not decompose into distinct subsets such that the simulation gets trapped in one or the others based on the particular random numbers used.
- The state of the simulation in the distant future must be effectively independent of the state in the past. In particular, the initial state at the start of a simulation replication must become irrelevant when simulating far enough into the future. This requires at least a minimal level of randomness and forgetfulness.

We now suppose that these conditions are satisfied, and the output process $Y_1, Y_2, \ldots$ within a replication is such that $Y_m \xrightarrow{D} Y$, a random variable Y whose distribution does not depend on the time index. We refer to Y as the steady state, and the existence of Y implies that limit properties such as $\mu = \mathrm{E}(Y)$ are well defined.

Even though Y_m has a steady state, the estimator $\bar{Y}(m)$ may still be biased (as in the $M/G/1$ queue). If we simulate long enough, then we want the bias in $\bar{Y}(m)$ to disappear. Define the *asymptotic bias* as

$$\begin{aligned}\beta &= \lim_{m\to\infty} m(\mathrm{E}(\bar{Y}(m)) - \mu) \\ &= \lim_{m\to\infty} m\left(\mathrm{E}\left(\frac{1}{m}\sum_{i=1}^{m} Y_i\right) - \mu\right) \\ &= \sum_{i=1}^{\infty}(\mathrm{E}(Y_i) - \mu) \qquad (5.4)\end{aligned}$$

which under some regularity conditions converges to a finite constant (Whitt, 2006). Thus, for large m, Bias $(\bar{Y}(m)) \approx \beta/m$. Estimating β is very difficult, so the fact that it must fall off quickly (as $1/m$) is important because we can overwhelm it with a long replication.

For any random variables $Y_1, Y_2, \ldots, Y_m$,

$$\mathrm{Var}\,(\bar{Y}(m)) = \frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\mathrm{Cov}(Y_i, Y_j). \tag{5.5}$$

This formula is a direct extension of the basic result $\mathrm{Var}(Y_1+Y_2) = \mathrm{Var}(Y_1) + \mathrm{Var}(Y_2) + 2\,\mathrm{Cov}(Y_1,Y_2) = \mathrm{Cov}(Y_1,Y_1) + \mathrm{Cov}(Y_2,Y_2) + \mathrm{Cov}(Y_1,Y_2) + \mathrm{Cov}(Y_2,Y_1)$. Notice that the expression has m^2 terms and we only have m observations, so there is no way to estimate it directly. What we hope, and can sometimes prove, is that the dependence structure of $Y_1, Y_2, \ldots$ also stabilizes for large m, not just the marginal properties such as mean and variance. The standard assumption is that beyond some point we can treat the process as being *covariance stationary*, meaning that for m large enough, $\sigma^2 = \mathrm{Var}(Y_m)$ and $\rho_k = \mathrm{Corr}(Y_m, Y_{m+k})$ are no longer a function of m. The quantity ρ_k is called the *autocorrelation* at lag k. For a covariance stationary process (5.5) simplifies to

$$\mathrm{Var}\,(\bar{Y}(m)) = \frac{\sigma^2}{m}\left(1 + 2\sum_{k=1}^{m-1}\left(1 - \frac{k}{m}\right)\rho_k\right). \tag{5.6}$$

Therefore, for a covariance stationary process the *asymptotic variance* is

$$\begin{aligned}\gamma^2 = \lim_{m\to\infty} m\mathrm{Var}\,(\bar{Y}(m)) &= \lim_{m\to\infty}\sigma^2\left(1 + 2\sum_{k=1}^{m-1}\left(1 - \frac{k}{m}\right)\rho_k\right) \\ &= \sigma^2\left(1 + 2\sum_{k=1}^{\infty}\rho_k\right). \end{aligned} \tag{5.7}$$

For the $\mathrm{Var}\,(\bar{Y}(m))$ to go to 0 with m we need $\gamma^2 < \infty$. Covariance stationarity alone does not guarantee this, since Y_i having "heavy tails" ($\sigma^2 = \infty$) or long-range dependence (ρ_k not falling off fast enough) could cause γ^2 to be infinite. When γ^2 exists then $\mathrm{Var}\,(\bar{Y}(m)) \approx \gamma^2/m$ for large m. Further, if ρ_k falls off quickly relative to m, then observations far enough apart are effectively uncorrelated and there is some hope of estimating γ^2 (see Chap. 8).

Bias is a systematic error, while variance reflects error due to random sampling. It might appear that they are equally important, because they both decrease as $1/m$. However, variance is not a measure of error, but of squared error: $\mathrm{Var}\,(\bar{Y}(m)) = \mathrm{E}\left[\{\bar{Y}(m) - \mathrm{E}\,(\bar{Y}(m))\}^2\right]$. A more relevant measure is the standard error, $\sqrt{\mathrm{Var}\,(\bar{Y}(m))}$, which decreases as $1/\sqrt{m}$. Thus, even though both are important, in a large-sample sense sampling variability dominates bias. A combined measure that treats bias and variance equally is the *mean squared error*,

$$\begin{aligned}\mathrm{MSE}\left(\bar{Y}(m)\right) = \mathrm{E}\left[(\bar{Y}(m)-\mu)^2\right] &= \mathrm{Bias}^2\left(\bar{Y}(m)\right) + \mathrm{Var}\left(\bar{Y}(m)\right) \\ &\approx \frac{\beta^2}{m^2} + \frac{\gamma^2}{m}.\end{aligned} \tag{5.8}$$

Remark 5.2. Although we defined modes of convergence in terms of a discrete-time process $Z_1, Z_2, \ldots$, there are completely analogous definitions for continuous-time processes, $\{Z(t), t \geq 0\}$ (e.g., the queue-length process $Y(t)$ instead of waiting time in the queue $Y_1, Y_2, \ldots$). In particular, the CLT for a continuous-time process shows that

$$\lim_{t\to\infty} \sqrt{t}\left(\frac{1}{t}\int_0^t Z(t)\,\mathrm{d}t - \mu\right) \xrightarrow{D} \gamma \mathrm{N}(0,1).$$

5.2.4 Beyond Sample Means

Not every performance measure is estimated by a sample mean (e.g., the variance). However, many estimators can be thought of as functions of sample means. A few key results are helpful for understanding the simulation literature.

Theorem 5.4 (Converging Together Lemma). *Suppose $Z_m \xrightarrow{D} Z$ and $W_m \xrightarrow{P} \eta$, where η is a constant. Then*

$$(Z_m, W_m) \xrightarrow{D} (Z, \eta).$$

This theorem states that if Z_m converges in distribution to Z, while W_m converges in probability to a constant, then no matter what their joint distribution is, they converge together to these limits. The converging together lemma is often paired with the following:

Theorem 5.5 (Continuous Mapping Theorem). *Suppose Z_m and Z are random variables in $\Re^d$; that $Z_m \xrightarrow{D} Z$; and that $h(\cdot)$ is a continuous function from $\Re^d$ to $\Re$. Then*

$$h(Z_m) \xrightarrow{D} h(Z).$$

This result states that when random variables converge in distribution, then functions of them may also converge. The statement of the theorem is actually stronger than necessary: h may have discontinuities in a set D as long as $\Pr\{Z \in D\} = 0$.

These two results turn the CLT into a practically useful tool. Suppose we have an estimator Z_n that satisfies the CLT

$$\sqrt{n}\,(Z_n - \mu) \xrightarrow{D} \gamma \mathrm{N}(0,1)$$

and we have an estimator $S_n^2 \xrightarrow{a.s.} \gamma^2$ (remember that almost sure convergence implies convergence in probability). Then if we let $h(a,b) = a/\sqrt{b}$, the converging together lemma and the continuous mapping theorem imply that

$$h\left(\sqrt{n}(Z_n - \mu), S_n^2\right) = \frac{\sqrt{n}(Z_n - \mu)}{S_n} \xrightarrow{D} \frac{\gamma \mathrm{N}(0,1)}{\gamma} = \mathrm{N}(0,1).$$

Thus, for large n we can argue that $Z_n \approx \mu + \mathrm{N}(0,1)\, S_n/\sqrt{n}$. This provides an asymptotic justification for the usual confidence interval $Z_n \pm 1.96 S_n/\sqrt{n}$, where 1.96 is the 0.975 quantile of the standard normal distribution. More generally, the converging together lemma and continuous mapping theorem are useful for establishing the validity of "plug-in estimators" where unknown constants (in this case the variance constant γ^2) are replaced with convergent estimators of them.

Here is another application of the continuous mapping theorem: Suppose that $Z_1, Z_2, \ldots$ are i.i.d. with finite mean μ and variance σ^2, and we want to estimate $h(\mu)$ for continuous and twice differentiable function h.

The continuous mapping theorem implies that $h(\bar{Z}(n))$ converges to $h(\mu)$. Since we will have to stop our simulation short of ∞, what can we say about the estimator $h(\bar{Z}(n))$? Using a Taylor series expansion

$$h(\bar{Z}(n)) \approx h(\mu) + h'(\mu)(\bar{Z}(n) - \mu)$$

so that provided $h'(\mu) \neq 0$

$$\begin{aligned}\sqrt{n}(h(\bar{Z}(n)) - h(\mu)) &\approx h'(\mu)\sqrt{n}(\bar{Z}(n) - \mu) \\ &\xrightarrow{D} h'(\mu)\sigma \mathrm{N}(0,1)\end{aligned}$$

by the CLT and continuous mapping theorem. Therefore, $\mathrm{Var}(h(\bar{Z}(n))) \approx h'(\mu)^2 \sigma^2/n$ for large n. Taking the Taylor series out to one more term, a similar argument shows that

$$\mathrm{Bias}(h(\bar{Z}(n))) = \mathrm{E}(h(\bar{Z}(n))) - h(\mu) \approx \frac{1}{2}\frac{h''(\mu)\sigma^2}{n}. \tag{5.9}$$

This approach for deriving the variance and bias of one statistic that is a function of another is usually called the "Delta method." The base statistic need not be a sample mean, but it does need to be a statistic whose asymptotic properties are known.

5.2.5 Simulation and Gaussian Processes

The theme of the previous section is that a simulation can be viewed as a stochastic process. Gaussian processes are a class of stochastic process that has been increasing useful as a way to represent various aspects of stochastic simulation design and analysis. Loosely speaking a Gaussian process (GP), also called a Gaussian random field, is a real-valued random function $G(\mathbf{x})$ on some domain $\mathbf{x} \in \mathscr{X}$ such for any finite k and $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k \in \mathscr{X}$, the values $(G(\mathbf{x}_1), G(\mathbf{x}_2), \ldots, G(\mathbf{x}_k))^\top$ have a multivariate normal (MVN) distribution. We briefly review the MVN distribution below, followed by a description of two useful classes of GPs, one in which $\mathbf{x}$ represents time, and a second in which it represents space (e.g., decision variables).

5.2.5.1 Multivariate Normal Distribution

Detailed descriptions of the MVN distribution can be found in many textbooks, including Johnson (1987). If $\mathbf{Z} = (Z_1, Z_2, \ldots, Z_k)^\top$ has a MVN distribution, we write $\mathbf{Z} \sim \mathrm{N}(\mathsf{m}, \mathsf{C})$, where m is the $k \times 1$ mean vector and C is the $k \times k$ variance matrix, with respective elements $\mathrm{E}(Z_i) = \mathsf{m}_i$ and $\mathrm{Cov}(Z_i, Z_j) = \mathsf{C}_{ij}$. Remember that $\mathrm{Cov}(Z_i, Z_i) = \mathrm{Var}(Z_i)$. A MVN distribution is fully specified by m and C, where C must be positive definite or positive semi-definite.

Let $\mathscr{A}, \mathscr{B}$ denote subsets of $\{1, 2, \ldots, k\}$ that may overlap, and need not be exhaustive. We use the notation $[\mathscr{A}]$ to indicate the rows of a vector with indices in $\mathscr{A}$, and $[\mathscr{A}, \mathscr{B}]$ to indicate the rows and columns of a matrix with indices in $\mathscr{A}$ and $\mathscr{B}$, respectively. For instance, if $\mathscr{A} = \{1, 2\}$ and $\mathscr{B} = \{2, 3\}$, then

$$\mathsf{m}[\mathscr{A}] = \begin{pmatrix} \mathsf{m}_1 \\ \mathsf{m}_2 \end{pmatrix} \text{ and } \mathsf{C}[\mathscr{A}, \mathscr{B}] = \begin{pmatrix} \mathsf{C}_{12} & \mathsf{C}_{13} \\ \mathsf{C}_{22} & \mathsf{C}_{23} \end{pmatrix}.$$

Two key properties of the MVN distribution are the following:

- If $\mathbf{Z} \sim \mathrm{N}(\mathsf{m}, \mathsf{C})$, then $\mathbf{Z}[\mathscr{A}] \sim \mathrm{N}(\mathsf{m}[\mathscr{A}], \mathsf{C}[\mathscr{A}, \mathscr{A}])$.
- If $\mathbf{Z} \sim \mathrm{N}(\mathsf{m}, \mathsf{C})$, then the conditional distribution of $\mathbf{Z}[\mathscr{A}]$ given $\mathbf{Z}[\mathscr{B}] = \mathbf{z}[\mathscr{B}]$ is also multivariate normal, with mean

$$\mathsf{m}[\mathscr{A}] + \mathsf{C}[\mathscr{A}, \mathscr{B}]\mathsf{C}[\mathscr{B}, \mathscr{B}]^{-1}(\mathbf{z}[\mathscr{B}] - \mathsf{m}[\mathscr{B}]) \tag{5.10}$$

  and variance

$$\mathsf{C}[\mathscr{A}, \mathscr{A}] - \mathsf{C}[\mathscr{A}, \mathscr{B}]\mathsf{C}[\mathscr{B}, \mathscr{B}]^{-1}\mathsf{C}[\mathscr{A}, \mathscr{B}]^\top. \tag{5.11}$$

  Thus, by observing $\mathbf{Z}[\mathscr{B}]$ one learns about $\mathbf{Z}[\mathscr{A}]$, as represented by a revised mean and a reduction of the variance.

5.2.5.2 Brownian Motion

The argument of Brownian motion (BM) typically corresponds to "time," so in this section we write $\{G(t); t \geq 0\}$ for standard BM, and let $\mathsf{m}(t) = \mathrm{E}[G(t)]$, $\mathsf{C}(t, t) = \mathrm{Var}[G(t)]$ and $\mathsf{C}(s, t) = \mathrm{Cov}[G(s), G(t)]$ where by convention $s \leq t$.

Standard BM is a one-dimensional random function that is everywhere continuous, but nowhere differentiable. Further,

$G(0) = 0$
$G(t) - G(s)$ is independent of $G(s)$

$G(t) - G(s) \sim \mathrm{N}(0, t-s)$
$C(s,t) = \min\{s,t\} = s.$

From standard BM we can construct $B(t) = \sigma G(t) + \mu t$, where the constant $-\infty < \mu < \infty$ is called the drift, and the constant $0 < \sigma < \infty$ is called the scaling. It follows immediately that $B(t) - B(s) \sim \mathrm{N}((t-s)\mu, (t-s)\sigma^2)$. BM has been studied extensively, including many results characterizing the probability of BM achieving large deviations from its mean.

There are several reasons why BM is important for stochastic simulation. Suppose that $Z_1, Z_2, \ldots$ are i.i.d. $\mathrm{N}(\mu, \sigma^2)$. Then the partial sum process $\{\sum_{i=1}^{n} Z_i; n = 1,2,\ldots\}$ is equal in distribution to $\{B(t); t = 1,2,\ldots\}$; that is, the process created by summing the first n i.i.d. normals has the same distribution as Brownian motion with drift μ and scaling σ evaluated at integer times. Thus, properties of BM may sometimes be used as approximations for the properties of simulation output processes.

Of course, simulation output processes need *not* be i.i.d. normal; nevertheless they may still behave like BM asymptotically. Suppose now that $Z_1, Z_2, \ldots, Z_n$ is a stationary process with marginal mean μ and variance σ^2, but is not necessarily either normally distributed nor independent. Define the standardized process on $0 \le t \le 1$:

$$Y_n(t) = \frac{\lfloor nt \rfloor (\bar{Z}(\lfloor nt \rfloor) - \mu)}{\sigma\sqrt{n}}. \tag{5.12}$$

Consider this process from two different perspectives: For fixed number n of Z's, $Y_n(t)$ is a random function defined for all $0 \le t \le 1$. For fixed $t \in (0,1]$, $Y_n(t)$ is a standardized average of a larger and larger number of Z's as n increases, just the sort of process to which the Central Limit Theorem applies. Under certain technical conditions (e.g., Glynn and Iglehart , 1990), the following Functional Central Limit Theorem (FCLT) holds:

Theorem 5.6. *As $n \to \infty$, $Y_n(t) \xrightarrow{\mathscr{D}} G(t)$ for all* $0 \le t \le 1$.

This is quite a powerful result: It states that the entire random function $Y_n(t)$ converges in distribution to standard BM, not just pointwise (for a given t) but jointly over all $0 \le t \le 1$. Thus, BM can be used as an approximation for more general simulation output processes than i.i.d. normal.

As an illustration, let $Z_i = 5 + 0.8(Z_{i-1} - 5) + X_i$ where the X_i are i.i.d. $\mathrm{N}(0,2)$; although the Z's are marginally normal, they are dependent. Figures 5.4 and 5.5 show one realization of $Z_1, Z_2, \ldots, Z_n$ and the standardized process $Y_n(t)$ for $n = 10$ and 1000, respectively.

Observant readers will notice that the plot of $Y_{1000}(t)$ seems too variable to be a standard BM: recall that $\mathrm{Var}[G(t)] = t$, but we observe that $Y_{1000}(t)$ deviates from 0 by as much as 4 in the plot. This behavior is an artifact of the dependence among the Z_i. The FCLT is a "limit theorem" as $n \to \infty$, and for this illustration a much larger n is required before $Y_n(t)$ is consistent with standard BM; Exercise 13 asks you to examine the effect of further increasing n.

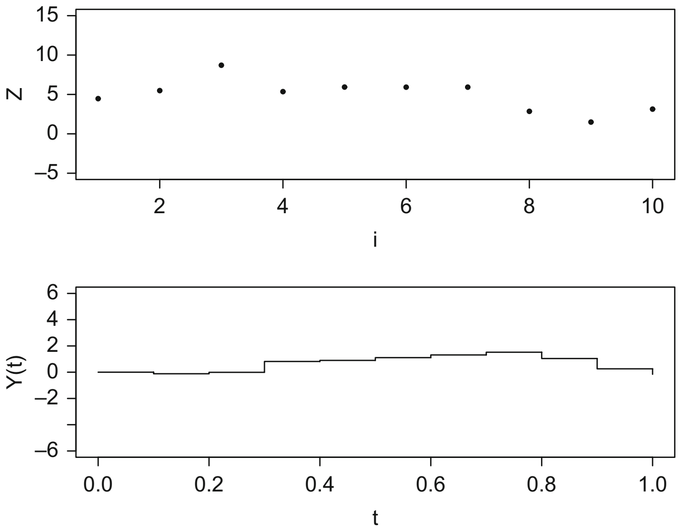

Fig. 5.4 Plot of $\{Z_1, Z_2, \ldots, Z_{10}\}$ and $Y_{10}(t)$ for $0 \leq t \leq 1$

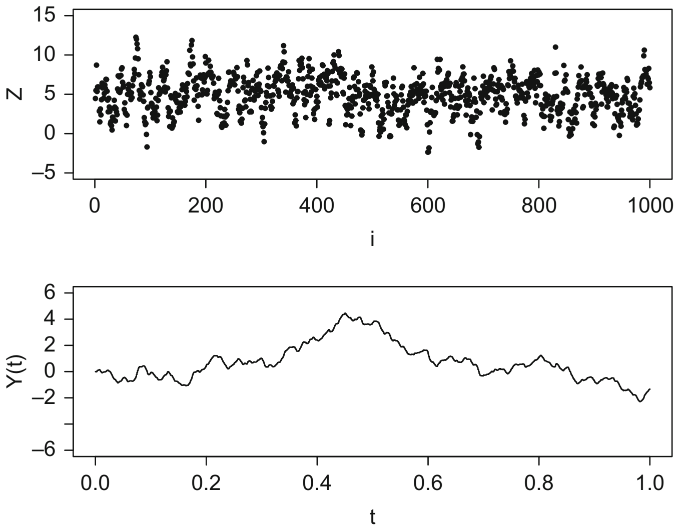

Fig. 5.5 Plot of $\{Z_1, Z_2, \ldots, Z_{1000}\}$ and $Y_{1000}(t)$ for $0 \leq t \leq 1$

5.2.5.3 Spatial Gaussian Processes

Suppose $g(\mathbf{x})$ is an unknown function, but we have the capability to evaluate it at a set of points $\mathscr{B} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$. Using these evaluations we would like to predict its value at any $\mathbf{x} \in \mathscr{X}$. A plausible property of $g(\cdot)$, if $g(\cdot)$ is to be predictable, is that when $\mathbf{x}$ and $\mathbf{x}'$ are "close" in the space $\mathscr{X}$, then the values of $g(\mathbf{x})$ and $g(\mathbf{x}')$ will be similar. The following is a clever way to exploit this property.

Since $g(\cdot)$ is unknown, model it as a realization of a GP $G(\cdot)$ defined on $\mathscr{X}$. Stated differently, $g(\cdot)$ is thought of as a random, but unobserved, instance of a GP. To specify the distribution of $G(\cdot)$ we need a mean function $\mathsf{m}(\mathbf{x}) = \mathrm{E}[G(\mathbf{x})]$ and a covariance function $\mathsf{C}(\mathbf{x}, \mathbf{x}') = \mathrm{Cov}[G(\mathbf{x}), G(\mathbf{x}')]$ for any $\mathbf{x}, \mathbf{x}' \in \mathscr{X}$. A common choice is to assume $\mathsf{m}(x) = \mathsf{m}$ for all $\mathbf{x}$, and $\mathsf{C}(\mathbf{x}, \mathbf{x}') = \mathsf{C}(d(\mathbf{x}, \mathbf{x}'))$, where $d(\cdot, \cdot)$ is a distance function; see Santner et al. (2003). The choice of covariance function is intimately connected to the properties of $g(\cdot)$, but loosely we want a function for which the covariance decreases as the distance between $\mathbf{x}$ and $\mathbf{x}'$ increases.

Having evaluated $\mathbf{g}[\mathscr{B}] = (g(\mathbf{x}_1), g(\mathbf{x}_2), \ldots, g(\mathbf{x}_k))^\top$, the natural predictor of $g(\mathbf{x})$ is the conditional mean of $G(\mathbf{x})$ given $\mathbf{G}[\mathscr{B}] = \mathbf{g}[\mathscr{B}]$, which from (5.10) is

$$\widehat{g}(\mathbf{x}) = \mathsf{m} + \mathsf{C}[\mathbf{x}, \mathscr{B}]\mathsf{C}[\mathscr{B}, \mathscr{B}]^{-1} (\mathbf{g}[\mathscr{B}] - \mathsf{m}\mathbf{1}_{k\times 1}) \,. \tag{5.13}$$

This highly effective method is called kriging, GP regression, or GP learning.

In stochastic simulation $g(\mathbf{x})$ might represent system performance as a function of controllable decision variables $\mathbf{x}$. For instance, $\mathbf{x}$ might be a staffing policy by hour of the day in a call center, and $g(\mathbf{x})$ might be the mean caller delay under this plan. What makes simulation different from the setting above is that $g(\mathbf{x})$ can only be *estimated*, rather than evaluated, by running a stochastic simulation. Stated more formally, the simulation generates $g(\mathbf{x}) + \varepsilon(\mathbf{x})$, where $\varepsilon(\mathbf{x})$ represents estimation error.

Stochastic kriging (Ankenman et al., 2010) represents this setting as $Y(\mathbf{x}) = G(\mathbf{x}) + \varepsilon(\mathbf{x})$, where $G(\cdot)$ and $\varepsilon(\cdot)$ are independent GPs, and derives statements about $G(\mathbf{x})$ conditional on observing $\mathbf{Y}[\mathscr{B}] = (Y(\mathbf{x}_1), Y(\mathbf{x}_2), \ldots, Y(\mathbf{x}_k))^\top = \mathbf{y}[\mathscr{B}]$. In particular, one can show that the conditional mean and variance are

$$\widehat{g}(\mathbf{x}) = \mathsf{m} + \mathsf{C}[\mathbf{x}, \mathscr{B}] \left\{\mathsf{C}[\mathscr{B}, \mathscr{B}] + \mathsf{C}_\varepsilon[\mathscr{B}, \mathscr{B}]\right\}^{-1} (\mathbf{y}[\mathscr{B}] - \mathsf{m}\mathbf{1}_{k\times 1}) \tag{5.14}$$

$$\mathrm{Var}[\widehat{g}(\mathbf{x})] = \mathsf{C}[\mathbf{x}, \mathbf{x}] - \mathsf{C}[\mathbf{x}, \mathscr{B}] \left\{\mathsf{C}[\mathscr{B}, \mathscr{B}] + \mathsf{C}_\varepsilon[\mathscr{B}, \mathscr{B}]\right\}^{-1} \mathsf{C}[\mathbf{x}, \mathscr{B}]^\top, \tag{5.15}$$

where C_ε is the covariance matrix of the estimation error. In practice there are many parameters in this result that need to be estimated, but we ignore that issue for now.

An important difference between kriging and stochastic kriging is the prediction at a point $\mathbf{x}_i \in \mathscr{B}$: In kriging the prediction is $g(\mathbf{x}_i)$, the observed value since there is no error. This is not the case for stochastic kriging because there is estimation error even at the simulated points.

5.3 Simulation as Computation

This book covers stochastic computer simulation with an emphasis on the modeling and statistical issues that arise due to it being "stochastic." In this section we consider the computational aspects of simulation, starting with a non-simulation computation to provide context.

Let $I = [0,1]$ be the unit interval and let be $g(\cdot)$ be an integrable function over I. For some nice functions g, the integral $\mu = \int_I g(x)\,dx$ can be expressed as $G(1) - G(0)$, where G is the anti-derivative of g that sometimes has an easy-to-evaluate form. However, "easy-to-evaluate form" is not the case for every integrable function, leading to the need for numerical approximations.

For instance, if we partition I into n subintervals $0 = x_0 < x_1 < \cdots < x_n = 1$, then the well-known trapezoid rule is

$$\int_I g(x)\,dx \approx \sum_{i=1}^{n} \frac{g(x_{i-1}) + g(x_i)}{2} \Delta x_i, \tag{5.16}$$

where $\Delta x_i = x_i - x_{i-1}$. Therefore, we can approximate the integral by evaluating g at only $n+1$ specific values of x. As n becomes larger and the Δx_i become smaller, the approximation becomes better *in theory*. However, when actually implemented on a computer, decreasing Δx_i can lead to a loss of numerical accuracy due to the finite representation of numbers; further, the computing effort to approximate the integral grows linearly in n.

Next suppose that $g(\mathbf{x})$ is a function of a d-dimensional $\mathbf{x}$, and $\mathbf{I}$ is the hyperbox $[0,1]^d$. Then using something like the trapezoid rule with n intervals in each dimension, the number of function evaluations to approximate the integral grows as n^d. Thus, there are computational limits on accurate, high-dimensional numerical integrations. And while there exist *much* better methods than the trapezoid rule (Süli & Mayers, 2003), the key observation that, to obtain an accurate approximation in higher and higher dimensions one has to bear substantially increased computation, still applies.

To connect numerical integration to stochastic simulation, we note that integrals can often be represented as expected values. For instance, if $\mathbf{X}$ is uniformly distributed[3] on $\mathbf{I}$, then

$$\mathrm{E}(g(\mathbf{X})) = \int_{\mathbf{I}} g(\mathbf{x})\,d\mathbf{x}. \tag{5.17}$$

This leads to the Monte Carlo estimator of μ:

$$\widehat{\mu} = \frac{1}{n} \sum_{j=1}^{n} g(\mathbf{X}_j), \tag{5.18}$$

[3] By "uniformly distributed" we mean that each dimension is uniformly distributed on $[0,1]$, and the dimensions are independent, so the joint density is $1^d = 1$ on $\mathbf{I}$.

where $\mathbf{X}_1, \mathbf{X}_2, \ldots$ are i.i.d. samples from the uniform distribution on $\mathbf{I}$. Notice that increasing n—the number of distribution samples—reduces the estimation error with no inherent loss of numerical accuracy since we are not slicing $\mathbf{I}$ into smaller and smaller hyperboxes. Further, the only impact that the dimension d has on the computational effort is in simulating $\mathbf{X}$, and that grows only linearly since we can simulate each dimension of $\mathbf{X}$ from an independent uniform distribution. For these reasons "Monte Carlo methods" are often superior to numerical methods for approximating high-dimensional integrals. Even though we have not to this point thought of our discrete-event stochastic simulation problems as the approximation of integrals, in many cases we could, and the same basic insights apply regardless.[4]

Although sometimes used interchangeably, the terms *accuracy* and *precision* are not equivalent, although both are related to error. Accuracy refers to how close a measurement is to the true value, while precision refers to how close measurements of the same item are to each other. Both are inversely proportional to their associated error. For instance, for the trapezoid rule with a particular choice of n and x_i

$$\text{accuracy} \propto \left| \int_I g(x)\,dx - \sum_{i=1}^{n} \frac{g(x_{i-1}) + g(x_i)}{2} \Delta x_i \right|^{-1}.$$

In the Monte Carlo context, accuracy is inversely proportional to the absolute bias of the estimator $|\mathrm{E}(\widehat{\mu}) - \mu|$; Monte Carlo estimators of integrals are often unbiased. Precision, on the other hand, is less relevant for numerical integration, but very relevant for Monte Carlo estimators: it is inversely proportional the $\mathrm{Var}(\widehat{\mu})$. MSE, as described in Sect. 5.2.3, encompasses both accuracy and precision for Monte Carlo estimators.

In pure numerical calculations, such as the trapezoid rule, additional computing effort is typically expended to increase accuracy; in stochastic simulation it is typically expended to increase precision (reduce variance), and sometimes to increase accuracy (reduce bias) as in steady-state simulation. *In a very real sense, the central topics of this book are assessing the MSE of simulation-based estimators and achieving an acceptable MSE through how we expend simulation effort.* This is not to say the discrete-event simulations are immune from numerical representation issues: For instance, random-number generators and random-variate generators may create or employ numerical values that are so large or so close to 0 that how they are handled can depend very much on the language in which they are coded or even the computer architecture itself. It is also possible that statistical summaries that are updated during execution can lose numerical precision if not updated carefully.

Since the typical discrete-event simulation generates a time-ordered sample path, it is inherently serial, moving from one event to the next one. For this reason event handling is often the primary computational bottleneck. By "event handling" we mean maintaining a time-ordered event calendar, inserting newly scheduled events in the appropriate place in the calendar, extracting the next event, and executing

[4] The exception is the Asian option example in Sects. 3.5 and 4.5, which has a lot in common with numerical integration and is not fundamentally a discrete-event simulation.

the event logic itself. The more scenarios (distinct systems) that need to be simulated, and the more replications of them that need to be executed, and the longer the simulated time within each replication, the more event handling that needs to be done. Clearly random-number generation, random-variate generation, updating statistical calculations, etc. also require expending computational effort, and they can in certain situations also be bottlenecks.

Simulations may also be run in various modes, especially when using commercial simulation products. For instance, such products can often produce a high-resolution graphical animation of model execution, and can generate an output trace of every event and update that happens during the run; both of these features create computation burdens, and in the case of animation it intentionally slows down the execution of events. The term "production runs" is used to indicate simulations that turn off animations, traces, and anything else that might slow execution so as to generate output measures as rapidly as possible. In this book we are typically doing production runs.

Certainly the most profound computational change for stochastic simulation since the graphical user interface is inexpensive parallel computing. Even low-end laptops are multi-core and have the capability of executing several processes in parallel. Commercial services such as Amazon Web Services, Microsoft® Azure and Google Cloud Platform rent parallel computing capacity to facilitate exploiting hundreds of processes at low cost.

At a high level, there are three ways to "parallelize" a discrete-event, stochastic simulation: (a) execute portions of the same simulation model in parallel with necessary synchronization, (b) execute distinct replications of the same simulation in parallel and concatenate the results, and (c) execute distinct simulation models or scenarios in parallel to compare the results.

Situation (a) is addressed by the PADS (Parallel and Distributed Simulation) community. Imagine simulating traffic in a large metropolitan area like Chicago, or the spread of a virus across the world. Although traffic can flow from anywhere in the Chicago area to any other, and a virus can spread from any country to any other, areas that are spatially distant may interact only weakly. Therefore, the simulation of distant regions can proceed largely in parallel but with some synchronization. This quite interesting approach will not be considered further here; see, for instance Fujimoto (2016).

Situations (b) and (c) are relevant to this text. To describe the statistical issues, suppose that we have $p+1$ parallel processes, which means we can execute $p+1$ "jobs" in parallel. We designate process $p+1$ as the "master" and the other p as "workers." In this set up, the master assigns simulation jobs to the workers, compiles results from the workers as they complete, and assigns further jobs to the workers. See Fig. 5.6. There are many possible computer architectures for parallel simulation, but this one is common and represents the key issues.

In situation (b) there is a single simulation model, the master assigns "simulate 1 replication" jobs to the workers and, to be specific, averages the returned outputs. It is immediately obvious that some care is needed in handling the pseudorandom numbers so that the replications use different portions of the sequence;

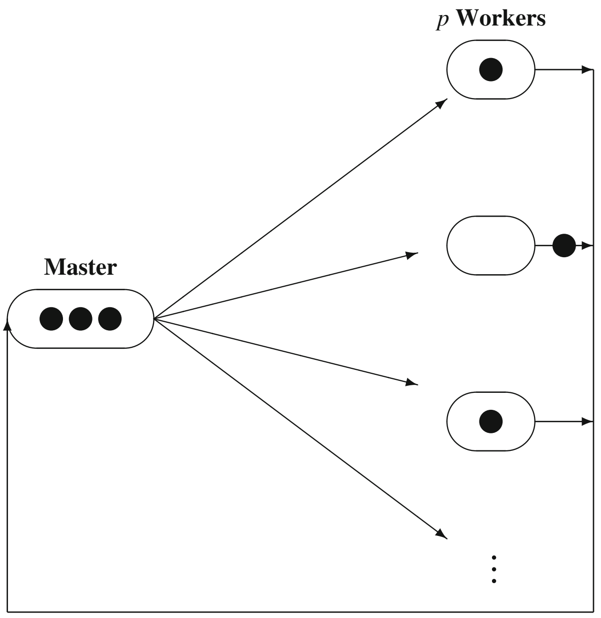

Fig. 5.6 Jobs represented as • being distributed to parallel processors to be returned when completed in a master-worker computing environment

see Sect. 6.5.3. However, even when that issue is addressed, Heidelberger (1988) and Glynn and Heidelberger (1991) observed that the outputs *received* by the master may not be i.i.d. if there is dependence between the clock (computer) time it takes to execute a replication and the value it returns. A queueing simulation of T units of *simulated* time provides an intuitive example: If on a particular replication the assigned pseudorandom numbers cause the system to be more congested than average—say because there are many more arrivals than average—then the clock time to execute that replication will be longer than average due to the larger number of arrival and service events that need to be handled. Thus, a performance measure like average queue length will have a positive dependence with execution time. Therefore, the master will tend to receive replications with smaller values of average queue length before those with larger values of average queue length, so the outputs are not identically distributed in the order they are received. As a result, the cumulative sample mean computed by the master will be biased. Exercise 20 asks you to derive the bias for a simple case. Of course, if the master needs a fixed number n of replications, makes no more than n assignments to workers, and then waits until all results are returned, then the problem disappears, but at the cost of forcing things to slow down.

In situation (c) there are many distinct models or scenarios that can be executed in parallel. This might occur when the goal is to optimize the performance of a system with respect to some controllable decision variables, and each distinct setting of these variable corresponds to a "scenario;" see Chap. 9. For instance, the goal might be to minimize the expected customer delay in a call or contact center by assigning agents to different work schedules that stay within a budget. For simplicity suppose that we do not simulate multiple replications of the *same* scenario in parallel, but do simulate *different* scenarios (perhaps for multiple replications) in parallel. Assuming that we pay for the parallel processing capability, a reasonable goal is to complete the optimization in a statistically valid way and as cost-effectively as possible.

Suppose that there are in total $K < \infty$ feasible solutions and $p \ll K$ parallel processors. A simple prescription such as "simulate all K models for 10 replications each and compute pairwise confidence intervals for the differences among them" now raises computational as well as statistical issues. Consider the following strategies:

- The master waits until all replications from all scenarios are complete and then performs the pairwise comparisons. This strategy idles the p workers while the master computes an excessive number of comparisons if K is large.
- As scenarios finish their replications, the master begins computing some of the pairwise comparisons while also dispatching unsimulated scenarios to the workers. The master must now do computations and keep track of which comparisons have been completed, and therefore may idle workers while being busy with these other tasks.
- The master dispatches roughly K/p scenarios to each worker to simulate and compare, and then exchanges results among them until all pairwise comparisons are complete. This strategy requires coordination among workers that finish their tasks out-of-sync, as well as a large quantity of data movement among the workers.

Clearly "all pairwise comparisons" is the simplest possible simulation optimization algorithm; more sophisticated algorithms will search for improving solutions and may eliminate solutions from further consideration with statistical guarantees. Without complete synchronization, which slows down progress, the statistical issues that arise in simulating a single scenario become even more complicated here; see Hunter and Nelson (2017) and Luo et al. (2015) as well as Chap. 9.

Appendix: Regenerative Processes and Steady State

How do we establish that a simulated system has a steady state? Or that simulation-based estimators satisfy a SLLN or CLT? As indicated earlier, this is a difficult question in general. In some cases we can argue that the simulation generating the output is a *regenerative process*. For deeper discussion beyond this appendix, see Haas (2002) and Glynn (2006).

Our goal is to prove that the output process Y_t has a steady state, and that the sample mean satisfies a SLLN and CLT. We will handle the continuous-time ($t \geq 0$) and discrete-time ($t = 0, 1, \ldots$) cases together.

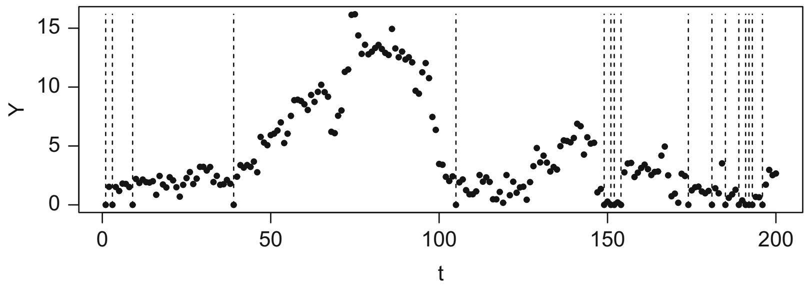

Fig. 5.7 Plot of successive waiting times in $M/G/1$ queue with times when customers arrive to an empty system indicated by dashed lines

Suppose that in the same simulation we can identify a renewal process $\{S_i, i = 0, 1, 2, \ldots\}$; by a renewal process we mean that

$$\begin{aligned} S_0 &= 0 \\ S_i &= A_1 + A_2 + \cdots + A_i, \end{aligned}$$

where A_i are i.i.d. with $\Pr\{A_i = 0\} < 1$ and $\Pr\{A_i < \infty\} = 1$. The output processes $\{Y_t; t \geq 0\}$ or $\{Y_t; t = 0, 1, \ldots\}$ are *regenerative processes* if taken together

$$\{Y_t; S_i \leq t < S_{i+1}\}, i = 0, 1, 2, \ldots$$

are i.i.d. Notice that the output process Y_t may be highly dependent; being regenerative means that the behavior of $\{Y_t; S_i \leq t < S_{i+1}\}$ and $\{Y_t; S_{i+1} \leq t < S_{i+2}\}$, say, are independent of each other, and have the same probability distribution. For this reason, regenerative processes are said to probabilistically "start over" at the renewal times $t = S_0, S_1, S_2, \ldots$.

For instance, an ergodic discrete or continuous-time Markov chain starts over every time it enters a fixed state, due to the Markov property. Many queueing systems start over probabilistically each time a customer arrives to find the system empty and idle. Figure 5.7 plots the waiting times Y_t of successive customers $t = 1, 2, \ldots$ arriving to an $M/G/1$ queue, with the vertical dashed lines indicating the customers who arrive to find the system empty and therefore have waiting time 0.

Let

$$Z_i = \int_{S_{i-1}}^{S_i} Y_t \,\mathrm{d}t$$

be the accumulated output over the ith renewal cycle (between the dashed lines). The integral becomes a sum for discrete-time output processes like Fig. 5.7. If Y_t is regenerative (with right-continuous sample paths if S_i is continuous valued), $\mathrm{E}(|Z_1|) < \infty$, $\mathrm{E}(A_1) < \infty$, and A_1 is aperiodic,[5] then the following hold:

- $Y_t \xrightarrow{D} Y$ (the process has a steady-state distribution);
- $\lim_{t\to\infty} t^{-1} \int_0^t Y_s \, \mathrm{d}s \xrightarrow{a.s.} \mathrm{E}(Z_1)/\mathrm{E}(A_1)$ (the sample mean converges to the expected total output per cycle divided by the expected length of a cycle); and
- $\mu = \mathrm{E}(Y) = \mathrm{E}(Z_1)/\mathrm{E}(A_1)$ (the steady-state mean is the almost sure limit of the sample mean).

Thus, regenerative structure establishes that there is a steady-state distribution and that as the length of the replication increases the sample mean satisfies a SLLN for the steady-state mean.

Now let $V_i = Z_i - \mu A_i$; this is the cumulative output from the ith cycle minus the predicted cumulative output based only on the length of the cycle. If it is also the case that $\mathrm{E}(V_1^2) < \infty$, then

- $\lim_{t\to\infty} \sqrt{t}(\bar{Y}_t - \mu) \xrightarrow{D} \gamma \mathrm{N}(0,1)$, where $\bar{Y}_t = t^{-1} \int_0^t Y_s \, \mathrm{d}s$; and
- The asymptotic variance $\gamma^2 = \mathrm{E}(V_1^2)/\mathrm{E}(A_1)$.

Therefore, the sample mean satisfies a CLT, and the asymptotic variance can be expressed in terms of properties of a regenerative cycle.

Establishing the existence of steady state using the regenerative argument amounts to identifying the renewal process S_i and arguing that $\{Y_t; S_i \le t < S_{i+1}\}$ are i.i.d. Typically $S_1, S_2, \ldots$ are times such that the state of the simulation is identical, and the probability distributions of all pending events are also the same. Further, one needs to establish that $\Pr\{A_i < \infty\} = 1$; that is, the time between regenerations is finite.

Consider the $M/G/1$ queue of Sect. 5.2.1 and sample path in Fig. 5.7. Let S_i be the ith time an arriving customer finds the system completely empty. At such times nothing that has happened in the past is relevant: there are no other customers in the system, the distribution of the time until the next arrival is exponential with parameter $\lambda = 1$, and the service time of the customer that just arrived is Erlang with mean $\tau = 0.8$ and three phases. The time between such occurrences is finite because $\lambda\tau < 1$, meaning that the system can keep up and therefore will be empty from time to time.

Exercises

1. Suppose that $Z_1, Z_2, \ldots, Z_n$ are i.i.d. with finite mean μ and variance σ^2. Use the WLLN and continuous mapping theorem to show that $S^2 \xrightarrow{P} \sigma^2$.
2. Look up and write down corresponding convergence definitions for continuous-time processes.

[5] Aperiodic means that for A_1 continuous valued, there is no $d > 0$ such that $\sum_{n=0}^{\infty} \Pr\{A_1 = nd\} = 1$; and for A_1 discrete valued, there is no integer $d \ge 2$ such that $\sum_{n=0}^{\infty} \Pr\{A_1 = nd\} = 1$.

3. Suppose that $Z_1, Z_2, \ldots, Z_n$ are i.i.d. with finite mean μ, and variance σ^2, and that $h(\cdot)$ is twice continuously differentiable at μ. Show that

$$n\left[\mathrm{E}\left(h\left(\bar{Z}(n)\right)\right) - h(\mu)\right] \longrightarrow \frac{1}{2}h''(\mu)\sigma^2.$$

4. Derive Eq. (5.5).
5. Starting with Eq. (5.5), show that for a covariance stationary process

$$\operatorname{Var}\left(\bar{Y}(m)\right) = \frac{\sigma^2}{m}\left(1 + 2\sum_{k=1}^{m-1}\left(1 - \frac{k}{m}\right)\rho_k\right).$$

6. Derive Eq. (5.9).
7. To prove that the results obtained by the Delta method are correct, conditions are needed on the higher-order terms in the Taylor series expansion. Derive or research these conditions.
8. In Sect. 5.2.4 it was stated that "The continuous mapping theorem implies that $h\left(\bar{Z}(n)\right)$ converges to $h(\mu)$." What mode of convergence does it satisfy? *Hint*: Convergence in distribution to a constant implies convergence in probability to that constant.
9. For the AR(1) process described in Sect. 3.3, derive the asymptotic MSE of the sample mean $\bar{Y}(m)$.
10. For the following scenarios, state whether the estimator will be biased and explain why or why not.

    a. For the stochastic activity network, we want to estimate the mean time to complete the project, and for our estimator we use Eq. (3.11).
    b. A European option is based on $X(T)$, the value of the asset at time T. To estimate the expected value of a European option and we use $\mathrm{e}^{-rT}(X(T) - K)^+$.
    c. For an $M/G/1$ queue, we are interested in estimating the mean waiting time of the tenth customer to arrive when the system starts empty, and we use Y_{10} from Eq. (3.3) as our estimator.
    d. Suppose that the parking lot represented by the $M(t)/M/\infty$ queue is open from 8 a.m. to 11 p.m. daily. A replication is defined by 1 day, and the garage starts empty each day. We want to estimate the expected value of the number of hours in a day when the garage has more than 2000 cars in it. We take as our output

$$Z = \int_0^T M(t)\,\mathrm{d}t,$$

where $T = 15$ h and

$$M(t) = \begin{cases} 1, & N(t) > 2{,}000 \\ 0, & N(t) \leq 2{,}000. \end{cases}$$

11. In addition to the AR(1), another surrogate model that is used to represent steady-state simulation output is the MA(1):
$$Y_i = \mu + \theta X_{i-1} + X_i,\ i = 1, 2, \ldots, m,$$
where $X_1, X_2, \ldots$ are i.i.d. $(0, \sigma^2)$ random variables, X_0 may be random or fixed, and $|\theta| < 1$. Show for this process that $\mathrm{Var}(Y_i) = (1+\theta^2)\sigma^2$, and
$$\mathrm{Cov}(Y_i, Y_{i+j}) = \begin{cases} \theta\sigma^2, & j = 1 \\ 0, & j > 1. \end{cases}$$
Use this information to derive the asymptotic MSE of $\bar{Y}(m) = \sum_{i=1}^{m} Y_i/m$ as an estimator of μ.
12. A mode of convergence not covered in this chapter is *convergence in mean square* (also called mean-square convergence). Look up the definition and describe how it relates to the other modes of convergence that were covered.
13. Recall the standardized average $Y_n(t)$ (Eq. (5.12) from Sect. 5.2.5.2) that converges to standard BM as $n \to \infty$, and the illustration using $Z_i = 5 + 0.8(Z_{i-1} - 5) + X_i$, where the X_i are i.i.d. N$(0, 2)$. The illustration is a specific instance of an AR(1) process $Z_i = \mu + \varphi(Z_{i-1} - \mu) + X_i$ where the X_i are N$(0, \sigma^2)$. Set $\mu = 0$ and $\sigma^2 = 1$, and run an experiment with $n = 100, 1000, 10{,}000$, and 100,000, and $\varphi \in \{0, 0.6, 0.9\}$. Notice that $\varphi = 0$ corresponds to standard BM observed discretely. As n increases do the standardized averages appear to be converging to BM for the other values of φ? Hint: to start the AR(1) in steady state, generate Z_0 from N$(0, \sigma^2/(1-\varphi^2))$.
14. Show that the kriging predictor (5.13) interpolates—meaning that $\widehat{g}(\mathbf{x}_i) = g(\mathbf{x}_i)$ for the observed points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k$—but the stochastic kriging predictor (5.14) does not in general. Hint: for the second part all you need is a counterexample.
15. Let $\ell(t) = \ln[\mathrm{E}(e^{tZ})]$. Then for random variables satisfying Cramér's Theorem, it can be shown that $I(z) = \sup_{t \in \Re}(tz - \ell(t))$, the Legendre transform of ℓ. Use this result to show that $I(z) = (z-\mu)^2/(2\sigma^2)$ for the normal distribution.
16. Prove that the Monte Carlo estimator (5.18) is unbiased for the integral.
17. In the integral $\int_{\mathbf{I}} g(\mathbf{x})\,d\mathbf{x}$, suppose that $\mathbf{I} = [a_i, b_i]^d$, where $a_i < b_i$ but not necessarily $[0, 1]$. Generalize the Monte Carlo method to this case. Hint: For any constant $v \neq 0$ we have $\int_{\mathbf{I}} g(\mathbf{x})\,d\mathbf{x} = v\int_{\mathbf{I}} g(\mathbf{x})/v\,d\mathbf{x}$
18. Estimate the integral $\int_0^4 (2x^2 - x)\,dx$ using the trapezoid rule with $n = 16$ intervals and using the Monte Carlo method using $n = 16$ samples; repeat the Monte Carlo method 10 times. Compare your results to the true value. For a simple one-dimensional integral like this is there any advantage to using Monte Carlo over numerical integration? Hint: You will need the result from the previous exercise.
19. We defined the Monte Carlo method for estimating $\int_{\mathbf{I}} g(\mathbf{x})\,d\mathbf{x}$ by simulating $\mathbf{X}$ from the uniform distribution over $\mathbf{I} = [0, 1]^d$. Suppose $f(\mathbf{x})$ is some other density over $\mathbf{I}$, and we simulate $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_n$ as i.i.d. from f instead. Show

that

$$\frac{1}{n}\sum_{j=1}^{n}\frac{g(\mathbf{X}_j)}{f(\mathbf{X}_j)}$$

has expected value $\int_{\mathbf{I}} g(\mathbf{x})\,d\mathbf{x}$. Comment: Choosing an f that reduces the variance leads to a better estimator for the same computational effort.

20. In a parallel simulation environment with one master and $p > 2$ workers, let Y_j be the value *returned* from the jth replication *requested* by the master, and let T_j be the corresponding execution (computer) time required to obtain it. Suppose that the output $Y \sim$ exponential with mean μ, and $T = Y$. Derive the $\mathrm{E}(Y_j), j = 1, 2, \ldots$. Hint: Remember that the first p simulation jobs will all start at time 0, and derive the $\mathrm{E}(Y_1)$.

Chapter 6
Simulation Input

This chapter covers simulation input, which includes the following:

Input modeling: Selecting (and perhaps fitting) the probability models that represent the uncertainty in the simulated system.
Random-variate generation: Representing the inputs as transformations of i.i.d. $U(0,1)$ random variables.
Random-number generation: Producing a good approximation to realizations of i.i.d. $U(0,1)$ random variates.

We present the topics in this order because input modeling is the objective, while random-variate generation and random-number generation are means to an end.

6.1 Input Modeling Overview

In many cases, the inputs themselves are not of great interest; what is of interest are the outputs that the inputs imply. For random-variate generation we can use provably correct methods; for random-number generation we can use generators with well-documented properties. Input modeling, on the other hand, may require judgement and is therefore open to more error. Since fitting distributions to data using methods such as maximum likelihood estimation (MLE) is a standard topic in statistics, our focus will be more on avoiding errors and less on the mechanics of fitting. We also discuss selecting input models when there are no data.

6.1.1 An Input Modeling Story

To motivate the important topics in input modeling, we will walk through a generic input modeling story, pausing from time to time to discuss it. The story represents a typical input modeling approach that makes standard assumptions; the discussion

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*, International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_6

outlines pitfalls and issues. The story parallels the simulation framework described in Sect. 5.1: There is a conceptual system of interest that does not yet exist, but there is a corresponding real system that is similar enough that some input data and system logic can be observed. The goal is to build a simulation of the conceptual system design.

Recall the hospital reception problem, Example 3.2: Under consideration was replacing the human receptionist by an electronic kiosk with a touch screen that might be a bit slower, or more variable, for patients and visitors who are less comfortable interacting with a touch screen. Therefore, the hospital management engineers wanted to evaluate how much additional delay that this might cause.

Hospital records were available to estimate the overall arrival rate. The physical basis for the Poisson distribution—arrivals from a large population of potential customers making independent decisions about when to arrive—suggests using it as the arrival-process input model. (For this reason exponentially distributed interarrival times are the default in some commercial simulation languages.)

To "fit" this input model, hospital records were used to count the number of arrivals over some time interval $[0,T]$ and the arrival rate λ was estimated by

$$\widehat{\lambda} = \frac{N(T)}{T},$$

where $N(t)$ is the number of arrivals that occurred by time t. This turns out to be an unbiased, maximum likelihood estimator of λ if the arrival process is really Poisson.

Notice that we are not just treating the arrival process as Poisson, but as *stationary* Poisson (constant arrival rate), which is a significantly stronger approximation. Unfortunately, if the arrival process was nonstationary Poisson, there is nothing about the estimator $\widehat{\lambda}$ to alert us; when we carry out this calculation we will get a number, even if the arrival rate is changing radically throughout the day as in the parking lot Example 3.1. *The only way to detect nonstationarity is to look at detailed arrival data, either the counts over smaller time intervals, or the interarrival times themselves if available.*

Suppose that the arrival rate is indeed constant throughout the day. Let T be the length of 1 day, and suppose we had recorded patient arrivals counts for, say, m days. Let $N_i(T), i = 1,2,\ldots,m$ be the counts from the m days. Then a statistical test could be used to see if the data support the hypothesis that the counts are Poisson. "Goodness-of-fit tests" are formulated as

H_0 : The chosen distribution with the estimated parameters is correct
H_1 : The chosen distribution with the estimated parameters is incorrect.

There are at least three reasons why the test might reject the Poisson input model:

1. The distribution of the data is substantially different from Poisson. For instance, if the reception desk only serves patients with appointments, and a fixed number of appointments are scheduled each day, then the counts are nearly identical and

therefore not Poisson. Also, scheduled appointments are typically spread evenly throughout the day and therefore arrivals are not as random as a Poisson model would predict.
2. The distribution of arrivals might differ by day of the week. For instance, Saturdays could be especially busy. Saturday arrival counts, considered by themselves, might be well-modeled as stationary Poisson, but not all days taken together. Thus, even if arrivals throughout a day can be treated as stationary Poisson there could still be nonstationarity across days.
3. We might have too much data. Why could this cause the hypothesis to be rejected? Treating the counts as Poisson is always an approximation if we are dealing with a physical (as opposed to simulated) system; real data do not come from probability distributions. Thus, any valid goodness-of-fit test will, if given enough data, reject every distribution. This does not mean that goodness-of-fit tests are useless, but it does mean that they should be used with caution. Goodness-of-fit tests make the most sense when there is a physical basis for the distribution choice, and we want a warning if the data vary substantially from that choice.

Hospital records would certainly provide counts. But if arrival times themselves are logged, then they would provide more detail, including interarrival times. If the arrival process is (stationary) Poisson, then the distribution of the interarrival times is exponential, and this fact provides another way to validate the input model.

The management engineers collected data on people interacting with the kiosk from a trial study with the vendor. For the service-time distribution, there is no strong process physics guiding the choice of distribution, as there was for the arrival process. Therefore, the management engineers analyzed the data using distribution fitting software and chose the distribution that the software recommends as the "best fit."

Distribution fitting software typically fits every relevant distribution it can using methods such as MLE, and then it recommends a candidate using some kind of score, often the best (smallest) goodness-of-fit statistic. Fitting all relevant distributions makes comparisons among the choices convenient, but otherwise this is a questionable approach. First, any score summarizes all aspects of the fit by a single number, without regard to where the lack of fit occurs, or if it matters. Second, if a goodness-of-fit test is to have any meaning at all, then there must be a distribution that was hypothesized due to some additional information (as we did with the Poisson choice for the arrival process); *there is no statistical meaning that can be attached to the distribution having the smallest test score from among an arbitrary collection.*

Are the data collected during the vendor trial representative of what will occur in the hospital? If the trial "patients and visitors" are vendor or hospital staff, then maybe not. Is learning anticipated? That is, are patients and visitors expected to get better at using the kiosk over time? If so, then any distribution fit to the trial data will not represent long-run behavior. *The message is that data have to be relevant to the situation at hand to be useful.*

Recall that the Pollaczek–Khinchine formula (Eq. (3.4)) gives the steady-state expected waiting time in an $M/G/1$ queue as

$$\mathrm{E}(Y) = \frac{\lambda(\sigma^2 + \tau^2)}{2(1 - \lambda\tau)},$$

where λ is the arrival rate of the Poisson arrival process and (τ, σ^2) are the mean and variance of the service-time distribution. One outcome of input modeling in this case is to establish values $\widehat{\lambda}, \widehat{\tau}$, and $\widehat{\sigma}^2$ for these parameters. Because this is an $M/G/1$ queue, it allows us to make a few observations:

- In this situation, at least in steady state, the particular service-time distribution chosen by the software does not matter *as long as it gets the mean and variance right*. Fortunately, it is often true—which is not the same as always true—that simulation results are not overly sensitive to the specific input distributions provided key characteristics are correct. In fact, we rely on this because *any* distribution we choose is an approximation of the physical process; there is no true distribution.
- Since our parameters are estimates, they will almost certainly be wrong. That is, $\widehat{\lambda} \neq \lambda$, $\widehat{\tau} \neq \tau$, and $\widehat{\sigma}^2 \neq \sigma^2$. While the expected waiting time is insensitive to the choice of service-time distribution, it is sensitive to the values of the mean and variance, τ and σ^2. Further, if $\widehat{\lambda}\widehat{\tau} > 1$, then there is no steady-state distribution of waiting time because the queue cannot keep up. While in reality the queue might be able to keep up, a sample of data could yield $\widehat{\lambda}\widehat{\tau} > 1$, particularly when $\widehat{\lambda}$ comes from one source (e.g., the hospital records), and $\widehat{\tau}$ comes from another (the vendor study).
- Although the distribution is not critical for the service process, it is for the arrival process: that is, the Pollaczek–Khinchine result does not depend on the particular service-time distribution except through its mean and variance, but it does depend on the arrival process being Poisson. If the arrival process is more or less variable than Poisson, then the congestion will be, respectively, more or less than what is predicted by the $M/G/1$ model.

What this "view through the queue" emphasizes is that the goal of input modeling is to get relevant outputs. This frees us from a futile search for the "true, correct distribution" and toward trying, as best we can, to approximate the information and data we have about the input processes.

6.1.2 Characteristics of Input Processes

It is difficult to formally define which random variables in a stochastic simulation are inputs, as opposed to outputs or intermediate random variables between inputs and outputs. Informally, inputs are stochastic processes whose (joint) distributions are treated as "known" in the simulation. Typically, inputs represent the uncertainty in the real world at its most basic level. We want inputs to have characteristics that match the real world, but we are usually not interested in those characteristics in and of themselves. As noted above, we want input models that provide relevant outputs.

We will divide input models into two classes:

Univariate input models: A univariate input random variable X is fully specified by its marginal distribution, F_X; when we need multiple realizations $X_1, X_2, \ldots$ then they are i.i.d. Examples of univariate inputs include the times to failure of several electronic components of the same type; the bed occupancy times of patients in a hospital; and the number of fans who arrive to a basketball game. We often model univariate inputs using parametric probability distributions, such as the Poisson, exponential, lognormal, and Weibull, with parameters that need to be tuned to the situation at hand. A particular parametric distribution may be justified by the physical basis of the input process or because it seems to provide a good fit to real-world data. We discuss selecting and fitting parametric input models in Sect. 6.2. Alternatively, we may simply reuse the data themselves; this is described in Sect. 6.2.4. When we do not have any relevant data, then other means for specifying F_X are required; see Sect. 6.2.6.

Multivariate input models: Multivariate input models define collections of random variables that are in some way related, as opposed to being i.i.d. realizations from a common distribution F_X. Examples of multivariate inputs include the quantity of whole milk, 2 % milk and skim milk ordered by a grocery store; the times to failure of four tires on the same car; the arrival times of customer calls to a call center; and the returns on each of the bonds held in a portfolio. A multivariate input model must directly or indirectly specify the marginal distribution of each component random variable, as well as the dependence among them. In principle, we could define the full joint probability distribution of the multivariate input, but this may be difficult in practice for the following reasons:

- The multivariate input is a (conceptually) infinite sequence, $S_1, S_2, S_3, \ldots$, which means that we cannot write a full joint distribution and need some other way to express how this sequence is constructed.
- We do not know the full joint distribution of, say (X_1, X_2, X_3), but we do know the marginals and have a partial characterization of the dependence among them, such as $\mathrm{Corr}(X_i, X_j)$ for all $i \neq j$.
- The marginal distributions (and perhaps the dependence as well) are a function of time; we refer such input processes as being *nonstationary*.

Multivariate input models for (possibly nonstationary) arrivals and for random vectors are described in Sect. 6.3.

6.2 Univariate Input Models

The situation addressed in this section often occurs in practice: We have a sample of real-world data, denoted $X_1, X_2, \ldots, X_m$, that we are willing to model as being i.i.d observations with a stable (unchanging) distribution F_X, and we want to generate random variates from F_X to drive our simulation. A standard approach is to somehow choose a parametric family of distributions to represent F_X and to fit the distribution to the data by estimating values of its parameters. Let $F(x;\theta)$ denote the parametric family with parameter vector θ. For instance, $F(x;\theta)$ might be the lognormal family with parameters $\theta = (\mu,\sigma^2)$, its mean and variance. Then "fitting" means using some appropriate estimator $\widehat{\theta} = \widehat{\theta}(X_1,X_2,\ldots,X_m)$, that is a function of the sample of real-world data, as the choice for θ.

The fact that we will use $F(x;\widehat{\theta})$ for variate generation is important in terms of how we think about evaluating the "goodness of fit." Let $\widehat{X}$ denote a random variable with distribution $F(x;\widehat{\theta})$. To be clear on this point, $\widehat{X}$ is not one of the real-world data values $X_1,X_2,\ldots,X_m$; instead, it is a random variate generated from the distribution $F(x;\widehat{\theta})$, where the parameters $\widehat{\theta}$ are functions of the real-world data.

What makes a good fit? There are two paradigms:

Inference: The classical paradigm is to hypothesize that there is a true but unknown distribution F_X, and it is a member of a parametric family $F(x;\theta)$. In this paradigm we try to discover the true F_X using a method that has good properties, averaged over the possible distributions and data we could see. Maximum likelihood estimators of θ arise from this paradigm, as does testing of the hypothesis H_0: $F(x;\widehat{\theta}) = F_X$. *In brief, the classical approach treats input modeling as a problem of statistical inference about the true distribution F_X, and employs methods that have good statistical properties.*

Matching: The matching paradigm is concerned with how closely the properties of $\widehat{X}$ match those of the real-world sample of data $X_1,X_2,\ldots,X_m$. For instance, it might be desirable to have

$$\mathrm{E}\left(\widehat{X}\,|X_1,X_2,\ldots,X_m\right) = \bar{X}.$$

Here the expectation is with respect to $F(x;\widehat{\theta})$. This condition indicates that the expected value of a random variable with distribution $F(x;\widehat{\theta})$ is the same as the sample mean of the data that was used to fit it. Depending on the application, there are many other properties of the data that it might be important to match, for instance certain tail percentiles. *In brief, the matching paradigm treats input modeling as capturing key features of the real-world data $X_1,X_2,\ldots,X_m$.*

In practice, univariate input modeling often combines aspects of the inference and matching approaches, so we will consider the balance between the two. We do not provide details about any estimation methods (such as maximum likelihood) or hypothesis tests (such as the Kolmogorov–Smirnov test) as these are well covered in basic statistics texts or other simulation references, such as Law (2007); also, they are implemented in input modeling software. Instead, we provide insight for proper use.

6.2.1 Inference About Univariate Distributions

Many of the well-known parametric families arise from particular process physics, often in some sort of limit. Perhaps the most well-known example is the normal distribution: If the random variable X is obtained by the summation of a large number of component random variables, then the distribution of X might tend to be normal by the central limit theorem. See Sect. 5.2.2 for the central limit theorem for the sum of i.i.d. random variables; there are also versions that apply to sums of non-identical and even dependent random variables (see for instance Lehmann (1999)).

For example, consider a worker who assembles laptop computers by hand. There are many steps in the assembly process, each of which may be modeled as a random variable. Because the total time to assemble a laptop is the sum of all of these component times, the nature of the process suggests that the total time to assemble a laptop might be well-modeled by a normal distribution.

Many discrete distributions are naturally defined by the process physics that they represent: For instance, the binomial distribution describes the number of successes in a fixed number of independent and identically distributed trials, while the negative binomial distribution describes the number of such trials required to achieve a given number of successes. Input modeling means deciding what constitutes a "trial" and a "success."

The inference paradigm is justified by having strong process physics that supports the choice of parametric family. In this section, we describe the theoretical support for selecting some common distributions (Poisson, lognormal, Weibull, and gamma) as input models. A comprehensive source for this type of insight is the series of books by Johnson, Kotz, and co-authors (Johnson et al., 2005, 1994, 1995, 1997; Kotz et al., 2000) and the website http://www.math.wm.edu/~leemis/chart/UDR/UDR.html.

6.2.1.1 The Poisson Distribution

An arrival process resulting from a large population of potential arrivals who make independent, but infrequent, decisions about when to arrive tends to be Poisson.

In a *renewal process*, the interarrival times between successive customers or entities, denoted $A_1, A_2, \ldots$, are i.i.d. nonnegative random variables with distribution G. Two stochastic processes that are derived from the A_n are

$$S_n = \begin{cases} 0, & n = 0 \\ \sum_{i=1}^{n} A_i, & n = 1, 2, \ldots \end{cases}$$
$$N(t) = \max\{n \geq 0 \colon S_n \leq t\}.$$

Thus, S_n is the time of the nth arrival, while $N(t)$ is the number of arrivals by time $t \geq 0$. The continuous-time process N is called the *arrival-counting process*.

Now consider a collection of independent renewal processes, with their associated arrival-counting processes

$$N_1(t), N_2(t), \ldots, N_h(t)$$

and each with their own interarrival-time distribution $G_i, i = 1, 2, \ldots, h$. Think of these as independent sources of arrivals, maybe different types of customers, or customers from different locations. The overall number of arrivals is the superposition arrival-counting process

$$C_h(t) = \sum_{i=1}^{h} N_i(t).$$

This is just the total number of arrivals from all sources by time t.

Now suppose that the number of independent arrival processes h is increasing, and consider

$$C_\infty(t) = \lim_{h \to \infty} \sum_{i=1}^{h} N_i(t).$$

We would expect this to blow up, but we impose two conditions on the interarrival distributions G_i as h increases:

$$\lim_{h \to \infty} \max_{i=1,2,\ldots,h} G_i(t) = 0 \tag{6.1}$$

$$\lim_{h \to \infty} \sum_{i=1}^{h} G_i(t) = \lambda t. \tag{6.2}$$

Recall that $G_i(t)$ is the probability that an interarrival time is less than or equal to t for the ith renewal arrival process; therefore, Condition (6.1) can be interpreted as implying that as we superpose more and more processes, the times between arrivals for each individual process are tending to become longer and longer. Stated differently, when the number of arrival sources h is large, the arrivals from each source are rare.

To interpret the second condition, let A_{1i} be the interarrival time from 0 until the first arrival for process i. Suppose that t is small, so that there is little chance of more than 1 arrival from each source. Then

$$\mathrm{E}[C_h(t)] \approx \mathrm{E}\left[\sum_{i=1}^{h} I(A_{1i} \le t)\right] = \sum_{i=1}^{h} G_i(t).$$

So Condition (6.2) can be interpreted as implying that the limiting arrival process has a stable arrival rate.

The following result shows that under these conditions, $C_h(t)$ will have a Poisson distribution as $h \to \infty$; that is, as we superpose more and more renewal arrival processes, the overall arrival process tends to be Poisson.

Theorem 6.1 (Poisson as Superposition of Renewal Processes). *If Condition (6.1) holds, then*

$$\lim_{h\to\infty} \Pr\{C_h(t) = j\} = \frac{\mathrm{e}^{-\lambda t}(\lambda t)^j}{j!},\ j = 0, 1, \ldots$$

if and only if Condition (6.2) holds.

A proof of an even more general version of this result can be found in Karlin and Taylor (1975).

Consider the people living in the Deerfield, Highland Park, and Northbrook communities north of Chicago who might call the local Subaru dealer to make an appointment for repair. Each owner's calls are infrequent. In addition, there is no reason to coordinate with other Subaru owners about when to call for service; thus, their calls are independent. The theorem suggests that the number of calls for appointments over any time period might be modeled as having a Poisson distribution. Notice, however, that if the Subaru dealer schedules a fixed number of repairs per day, then the number of actual arrivals (as opposed to calls) to the dealer will not be Poisson since a bound on appointments imposes dependence on the arrival process.

6.2.1.2 The Lognormal Distribution

A random variable formed as the product (multiplication) of independent, positive random variables tends to have a lognormal distribution.

By definition, if a random variable X is such that $\ln(X)$ has a normal distribution, then X has a lognormal distribution. This connection gives rise to one asymptotic justification for selecting the lognormal distribution as an input model: Suppose that $Z_1, Z_2, \ldots$ are i.i.d. positive random variables, and

$$X = \prod_{i=1}^{m} Z_i.$$

Then

$$W = \ln(X) = \sum_{i=1}^{m} \ln(Z_i).$$

Thus, if $\mathrm{E}\left(\ln(Z_1)^2\right) < \infty$, then the central limit theorem establishes that

$$\sqrt{m}\left(\frac{1}{m}\sum_{i=1}^{m} \ln(Z_i)\right) = \frac{W}{\sqrt{m}}$$

is asymptotically normally distributed as $m \to \infty$. Then since $X = \exp(W)$, the continuous mapping theorem can be used to show that X is asymptotically lognormal. And just as the condition of being identically distributed can sometimes be relaxed for the central limit theorem, the Z_i are not required to be identically distributed for the limiting distribution to be lognormal.

A context in which the lognormal distribution arises as a natural input model is in returns on investments that are compounded. Suppose that $Z_i = (1 + R_i)$, where the R_i are i.i.d. and > -1. Here Z_i represents the rate of return in the ith period, so the overall rate of return for m periods is $X = \prod_{i=1}^{m} Z_i$. Thus, the value of an initial investment of \$$S$ has value $S \cdot X$ after m periods, which might be modeled as lognormally distributed if m is large.

6.2.1.3 The Weibull and Gamma Distributions

The Weibull and gamma distributions often arise as models of system reliability (time until failure).

There are several equivalent parameterizations of the Weibull distribution, but this is a common one for the pdf and cdf, respectively:

$$f_T(t) = \alpha\beta^{-\alpha}t^{\alpha-1}\mathrm{e}^{-(t/\beta)^{\alpha}} \tag{6.3}$$

$$F_T(t) = 1 - \mathrm{e}^{-(t/\beta)^{\alpha}} \tag{6.4}$$

for $t \geq 0$, where $\alpha > 0$ is called the shape parameter, and $\beta > 0$ is the scale parameter. Notice that when $\alpha = 1$ the Weibull distribution becomes the exponential distribution with mean β.

Let T represent the time until failure of a system. The hazard function of T is a useful way to think about how a distribution represents reliability. The definition of the hazard function is

$$h(t) = \frac{f_T(t)}{1 - F_T(t)} \tag{6.5}$$

which can be interpreted as the system failure rate at time t (numerator), given survival to time t (denominator). For the Weibull,

$$h(t) = \alpha\beta^{-\alpha}t^{\alpha-1}$$

for $t \geq 0$, which is a constant $1/\beta$ if $\alpha = 1$.

By taking the derivative of $h(t)$ with respect to t, we can show that $h(t)$ for the Weibull is increasing for $\alpha > 1$, decreasing for $\alpha < 1$, and constant for $\alpha = 1$. An increasing hazard is appropriate for a system that is more likely to fail as it ages; a decreasing hazard is appropriate for a system that is subject to early failure; while a constant hazard (the exponential special case) means that the failure rate is unaffected by age.

The gamma distribution is similar in many ways to the Weibull; the pdf is

$$f_T(t) = \frac{\beta^{-\alpha}t^{\alpha-1}\mathrm{e}^{-(t/\beta)}}{\Gamma(\alpha)} \tag{6.6}$$

for $t \geq 0$, where $\Gamma(\alpha) = \int_0^\infty t^{\alpha-1}\mathrm{e}^{-t}\,\mathrm{d}t$ is the gamma function. The parameters α and β have the same interpretation as for the Weibull, with $\alpha = 1$ again giving the exponential distribution with mean β. Neither the cdf nor the hazard function of the gamma distribution has a simple closed form, but it can be shown that the hazard function may be increasing, decreasing, or constant, just as with the Weibull. So how might an understanding of the physics of a process lead to a sensible choice between these two distributions?

Displayed below are the pdfs with scale parameter $\beta = 1$:

Weibull	gamma
$\alpha t^{\alpha-1}\mathrm{e}^{-t^{\alpha}}$	$\Gamma(\alpha)^{-1}t^{\alpha-1}\mathrm{e}^{-t}$

The similarities are obvious. But notice that as t increases (t represents the time of failure), the tail of the gamma distribution becomes that of an exponential distribution, no matter what the choice of the shape parameter α. Recall that the hazard function for the exponential is constant. Thus, as the item of interest ages without failing, it eventually has a constant failure rate if it is modeled as a gamma distribution. For the Weibull distribution, however, the shape parameter affects the tail; in particular for $\alpha > 1$ the tail is lighter than exponential, indicating an increasing failure rate even for older items.

Consider the following two examples:[1] A new computer operating system is released to the market. Which of the Weibull or gamma distributions might be an appropriate model for the time to the first failure of the operating system? In this case, a probability model would likely be one that has increased risk of a failure initially, but after the software gets "broken in" the risk of a failure decreases to a constant value. Since the risk is constant, one would like to use a gamma distribution, rather than a Weibull distribution, because of the exponential right-tail property. Notice that if we used the Weibull distribution with $\alpha = 1$, then we would have constant failure rate even at product release, which is clearly not appropriate.

As a second example, consider the time to failure of a ball bearing in a machine. In this case, since the ball bearing is physically wearing out, it is not reasonable that as time increases the risk of failure becomes constant. The right-hand tail of a gamma distribution is too heavy, and the Weibull distribution is the more appropriate model.

6.2.2 Estimation and Testing

When there is a strong physical basis for choosing a parametric family of distributions, then parameter estimation with a method like maximum likelihood, and goodness-of-fit hypothesis testing of the choice, make sense. These topics are well covered in many textbooks, including Law (2007), and both parameter estimation and testing are included in input modeling software packages. Therefore, this section provides a few comments to put estimation and testing in context.

[1] These examples were suggested in a personal communication by Dr. Lawrence Leemis of the William & Mary.

6.2.2.1 Maximum Likelihood Estimation (MLE)

Suppose that we have a sample of real-world data $X_1, X_2, \ldots, X_m$ and we have a strong physical basis for modeling it as being an i.i.d. sample from a distribution with pdf $f^\star(x;\theta)$, where θ is a vector of unknown parameters. Then the likelihood function is defined to be

$$L(\theta) = \prod_{i=1}^{m} f^\star(X_i;\theta). \tag{6.7}$$

The MLE is the value of θ that maximizes $L(\theta)$ given the available data:

$$\widehat{\theta} = \mathrm{argmax}_{\theta} L(\theta).$$

MLEs have many good statistical properties under the assumption that $f^\star(x;\theta)$ is the true, correct distribution. The only reasonable justification for this assumption is that there is strong process physics supporting the selection of $f^\star(x;\theta)$.

Notice also that the likelihood function (6.7) is formed by assuming that the data are independent. Even if $f^\star(x;\theta)$ is an appropriate parametric family of distributions for X, (6.7) is not the likelihood function if the data are dependent (in which case the likelihood function involves the full joint distribution). Preliminary examination of the data for signs of dependence is always worthwhile.

6.2.2.2 Goodness-of-Fit Testing

The premise behind goodness-of-fit testing is that there is a true, correct distribution $f^\star$ with true, correct parameter values $\theta^\star$. The hypothesis test is then

$$\begin{aligned} H_0&: f(x;\widehat{\theta}) = f^\star(x;\theta^\star) \\ H_1&: f(x;\widehat{\theta}) \neq f^\star(x;\theta^\star). \end{aligned}$$

Because real-world data do not come from probability distributions, we know before executing the test that the null hypothesis is false. However, if we have a strong basis for choosing $f(x;\theta)$ and have used an appropriate parameter estimator $\widehat{\theta}$, then such a test is useful because it may alert us to egregious departures of the data from the proposed input model. A few additional points are worth keeping in mind:

- When the sample size is small, goodness-of-fit tests often have low power and thus may accept (formally, fail to reject) many possible parametric families of distributions. As discussed above, the Weibull and gamma families are very similar, and therefore may be indistinguishable when m is small. Thus, accepting the null hypothesis should not be interpreted as proof that the correct, or best, choice has been made.

- When the sample size is large, goodness-of-fit tests will often reject every choice of parametric distribution family because real-world data do not come from probability distributions, and when m is large, then the test has enough power to recognize this. Thus, rejecting the null hypothesis does not necessarily mean that the chosen distribution should be discarded.
- Rather than report an accept/reject decision, most input modeling software provides either the value of the test statistic itself (a small value is better for most tests), or a p-value. The p-value is the Type I error level at which one would just reject the null hypothesis. Thus, a large p-value (say >0.1) suggests accepting the chosen distribution rather than taking a large risk by rejecting it.
- There are many goodness-of-fit tests, each sensitive to different sorts of departures from the null hypothesis. Thus, it is entirely possible that one test may give a large p-value (suggesting that you accept the chosen distribution), and another may give a small p-value (suggesting you reject the chosen distribution) for the same data. In fact, the very popular chi-squared test may give both conclusions depending upon how many bins into which the data are divided. *For this reason, automated selection of an input model by fitting many of them and choosing the one with the smallest value of a particular test statistic or the largest p-value for that test seems questionable.* If there is no strong physical support for a particular distribution, then the matching paradigm (discussed below) may be more relevant.

Related to goodness-of-fit tests are "information criteria." Two popular statistics are the Akaike information criterion (AIC) and Bayesian information criterion (BIC). Both rank model fits based on a penalized likelihood function. There is a stronger argument for ranking the fit of several distributions to the same data using AIC or BIC than there is for comparing the p-values of a test; however, neither directly answers the question of whether *any* of the proposed distributions is a good choice.

Goodness-of-fit tests and information criteria represent fit or lack of fit by a summary statistic. Distribution fitting software usually also contains graphical tools for visually evaluating fit, and these give a more comprehensive picture. The most popular is the density-histogram plot, but your perception of fit for that plot can be influenced by how the data are grouped to form the histogram. A better tool that does not require grouping is the quantile–quantile (Q-Q) plot. For a random variable with continuous, increasing cdf F_X its q-quantile (for $0 < q < 1$) is $F_X^{-1}(q)$, the inverse cdf evaluated at q. Recall that we used the inverse cdf for variate generation, $X = F_X^{-1}(U)$ with $U \sim U(0,1)$. The Q-Q plot plots the sorted input data $X_{(1)} \le X_{(2)} \le \cdots \le X_{(m)}$ against the fitted inverse cdf $\widehat{F}^{-1}$ at what might be called a "perfectly generated sample"

$$\widehat{F}^{-1}\left(\frac{0.5}{m}\right) < \widehat{F}^{-1}\left(\frac{1.5}{m}\right) < \cdots < \widehat{F}^{-1}\left(\frac{m-0.5}{m}\right).$$

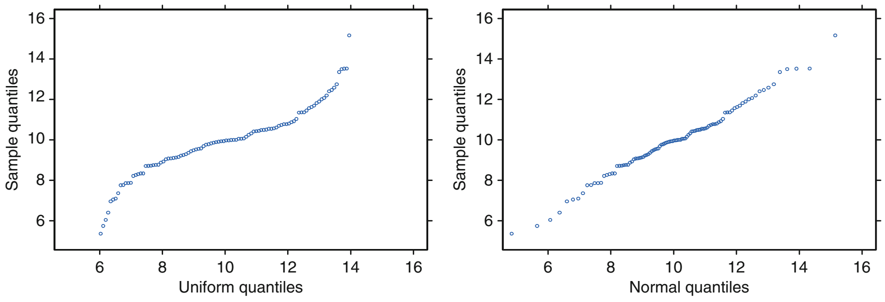

Fig. 6.1 Q-Q plot resulting from fitting a uniform (*left*) and normal (*right*) distribution to a set of input data

Stated more succinctly, it plots $X_{(i)}$ vs. $\widehat{F}^{-1}\left((i-0.5)/m\right)$, and a good fit is indicated by $X_{(i)} \approx \widehat{F}^{-1}\left((i-0.5)/m\right)$, $i = 1,2,\ldots,m$. Deviation from an approximate 45° line indicates not only lack of fit, but also where it occurs.

To illustrate how it works, Fig. 6.1 shows Q-Q plots for a set of data that actually are normally distributed, but for which both uniform and normal distributions have been fit. The uniform distribution is far too heavy in its tails, which the plot reveals in the way that the tails deviate more severely from a line.

6.2.2.3 Known and Unknown Bounds

Many standard distributions have their support on a fixed domain; for instance, the lognormal, Weibull, and gamma distributions are defined on $[0,\infty)$, while the beta distribution is defined on $[0,1]$.

A random variable X' describing outcomes on $[0,\infty)$ can be shifted to $[a,\infty)$ simply by adding a constant: $X = a + X'$. This changes the mean but not the variance. A random variable Y' describing outcomes on $[0,1]$ can be shifted and scaled to $[a,b]$ by the transformation $Y = a + (b-a)Y'$; this changes both the mean and variance.

Obviously these transformations can be inverted to convert a random variable to a standard domain: $X' = X - a$ and $Y' = (Y-a)/(b-a)$. This is important if we encounter a situation in which there are known bounds. For instance, we might want to model a processing time using a distribution with standard domain $[0,\infty)$, but we know that the process cannot take less than 4 min. To fit the observed data $X_1', X_2', \ldots, X_m'$, we should first transform it by subtracting 4 (e.g., $X_i = X_i' - 4$), fit the distribution to the transformed data, and then add 4 to each random variate generated.

Using known bounds is not the same as trying to infer an unknown bound. Distribution fitting software will often treat each distribution on $[0,\infty)$ as having an additional lower-bound parameter a, and each distribution on $[0,1]$ as having two additional upper and lower-bound parameters a,b; these additional parameters are then included in the fitting. Sometimes the bounds are estimated heuristically (e.g.,

$\widehat{a}$ is the smallest observation), and sometimes they are included in the likelihood function. *Known bounds should typically be used instead of estimated bounds, if available*. A known bound brings in additional information that may not be present in the data, and estimated bounds may substantially over (under) estimate lower (upper) bounds, particularly in small samples. That said, loose bounds that are not actually attainable by the process or represented in the data can wreck havoc on distribution fitting software by forcing the distribution to stretch to the bound. Unless the bound is really "known" it is often better to estimate the bound so it is consistent with the data.

6.2.3 Matching Properties of Univariate Distributions

When there is not a strong physical basis for choosing a particular family of distributions, then an alternative to trying to infer the "true" distribution is to use a very flexible family of distributions and obtain values for its parameters so that it closely matches properties of the data. In this section we describe two common matching methods and present one flexible family.

Recall that for the $M/G/1$ queue the steady-state expected waiting time depends only on the mean and variance of the service-time distribution; thus, any nonnegative distribution that matches the correct mean and variance will give the right simulation results. Typically more than just the mean and variance matter, which is why there are many two-parameter distributions (Weibull, gamma, lognormal, etc.) from which to choose. Fortunately, it is often the case that if we can match the first four moments (defined below) of the input random variable correctly, then the particular family of distributions is not as important.[2]

Let X be a random variable with distribution F_X. The *kth moment* of X is

$$\mathrm{E}(X^k),\ k = 1,2,\ldots.$$

Of more use for input modeling are the standardized central moments, specifically the first four:

$$\begin{array}{ll} \mu_X = \mathrm{E}(X) & \text{mean} \\ \sigma_X^2 = \mathrm{E}[(X-\mu_X)^2] & \text{variance} \\ \alpha_3 = \mathrm{E}[(X-\mu_X)^3]/\sigma_X^3 & \text{skewness} \\ \alpha_4 = \mathrm{E}[(X-\mu_X)^4]/\sigma_X^4 & \text{kurtosis} \end{array} \tag{6.8}$$

The skewness is a measure of symmetry (symmetric distributions have $\alpha_3 = 0$), while the kurtosis is a measure of tail weight (the normal distribution has kurtosis $\alpha_4 = 3$; sometimes $\alpha_4 - 3$ is called the excess kurtosis). If α_3 and α_4 are finite, then $\alpha_4 > 1 + \alpha_3^2$ and the inequality is strict. This relationship defines a plane of feasible (α_3^2, α_4) values.

[2] The most common exceptions are when system performance depends critically on the behavior of the extreme tail of the distribution, or when one or more of the first four moments are infinite.

Suppose that the first four central moments of X, $(\mu_X, \sigma_X^2, \alpha_3, \alpha_4)$, exist. Let μ and σ^2 be a desired mean and variance, and define a new random variable

$$X' = \mu + \sigma \left(\frac{X - \mu_X}{\sigma_X} \right). \tag{6.9}$$

Then it is easy to show that X' has mean μ and variance σ^2; in the exercises you are asked to show that X and X' have the same skewness and kurtosis. Thus, any random variable X can be transformed to match a desired mean and variance without altering its skewness and kurtosis. In matching, α_3 and α_4 are the challenges.

For a parametric distribution $F(\cdot;\theta)$, its central moments are functions of θ; denote these as $\mu(\theta), \sigma^2(\theta), \alpha_3(\theta)$, and $\alpha_4(\theta)$, respectively. We call a parametric distribution "flexible" if its parameters θ allow coverage of a substantial portion of the feasible (α_3^2, α_4) plane. As (6.9) shows, the mean and variance of F are not critical since we can scale and shift any distribution to have the desired first two central moments.

Many well-known distributions provide little flexibility; for instance the normal distribution has $\alpha_3 = 0$ and $\alpha_4 = 3$; the exponential distribution has $\alpha_3 = 2$ and $\alpha_4 = 9$; while the gamma distribution (6.6) has a little flexibility with $\alpha_3 = 2/\sqrt{\alpha}$ and $\alpha_4 = 3 + 6/\alpha$. We describe some more flexible distributions below.

The sample standardized central moments of the data are

$$\begin{aligned}
\bar{X} &= \frac{1}{m}\sum_{i=1}^{m} X_i \\
\widehat{\sigma}^2 &= \frac{1}{m}\sum_{i=1}^{m} (X_i - \bar{X})^2 \\
\widehat{\alpha}_3 &= \frac{1}{m}\sum_{i=1}^{m} (X_i - \bar{X})^3/\widehat{\sigma}^3 \\
\widehat{\alpha}_4 &= \frac{1}{m}\sum_{i=1}^{m} (X_i - \bar{X})^4/\widehat{\sigma}^4.
\end{aligned}$$

To match the moments, we solve the system of equations

$$\begin{aligned}
\mu(\theta) &= \bar{X} \\
\sigma^2(\theta) &= \widehat{\sigma}^2 \\
\alpha_3(\theta) &= \widehat{\alpha}_3 \\
\alpha_4(\theta) &= \widehat{\alpha}_4
\end{aligned}$$

for θ; denote the solution by $\widehat{\theta}_M$. Unless we bring in some other type of information, we need at least four parameters to match four moments.

Provided that the parametric family is flexible enough to match the moments exactly, then using $F(\cdot;\widehat{\theta}_M)$ as the input model guarantees that the simulation input has the same central moments as the observed data $X_1, X_2, \ldots, X_m$; that is, they match. For this reason the "method of moments" is an appealing approach that is widely used. Notice that one need not always fit four moments. For instance, the gamma distribution has $\theta = (\alpha, \beta)$ with $\mu(\theta) = \alpha\beta$ and $\sigma^2(\theta) = \alpha\beta^2$, so it can be fit by solving $\widehat{\alpha}\widehat{\beta} = \bar{X}$ and $\widehat{\alpha}\widehat{\beta}^2 = \widehat{\sigma}^2$.

A drawback of the method of moments is that there is no guarantee that the fitted distribution $F(\cdot; \widehat{\theta}_M)$ resembles the empirical distribution of the data. The next approach tries to achieve a closer match between these two.

A second matching approach exploits the following fact: If X is a random variable with strictly increasing cdf F_X, then $U = F_X(X)$ has a $U(0,1)$ distribution. Exercise 36 asks you to show that this is indeed the case, but it seems reasonable since it reverses the inverse cdf method of variate generation. This fact implies that if $X_1, X_2, \ldots, X_m$ are i.i.d. with continuous distribution F_X, then $U_i = F_X(X_i), i = 1, 2, \ldots, m$ are i.i.d. $U(0,1)$. Going one step further, let $X_{(1)} \le X_{(2)} \le \cdots \le X_{(m)}$ be the order statistics of the sample of data, and let $U_{(i)} = F_X(X_{(i)}), i = 1, 2, \ldots, m$. Then $U_{(i)}$ has the distribution of the ith order statistic of m i.i.d. $U(0,1)$ random variables, for which it is known that

$$\mathrm{E}(U_{(i)}) = \frac{i}{m+1}.$$

If we want to match the behavior of $X_1, X_2, \ldots, X_m$ with a parametric distribution $F(\cdot; \theta)$, then another approach that is different from the method of moments is to set

$$\widehat{\theta}_U = \mathrm{argmin}_{\theta} \sum_{i=1}^{m} w(i) \left(F(X_{(i)}; \theta) - \frac{i}{m+1} \right)^2, \tag{6.10}$$

where $w(i)$ is a positive weight. The weights can be chosen based on a number of criteria; common choices are $w(i) = 1$ and $w(i) = (m+2)(m+1)^2/[i(m-i+1)]$. The latter weight is $1/\mathrm{Var}(U_{(i)})$ so that it gives order statistics with smaller variance more weight.

What does this fit achieve? Notice that if the fit was perfect, then

$$F(X_{(i)}; \widehat{\theta}_U) = \frac{i}{m+1}.$$

Compare this with the empirical cumulative probability of the real-world data:

$$\frac{\#\{X_j \le X_{(i)}\}}{m} = \frac{i}{m}.$$

Thus, with a perfect fit the cumulative probabilities of the fitted distribution at the data points $F(X_i; \widehat{\theta}_U), i = 1, 2, \ldots, m$ are nearly the same as the cumulative probabilities of the data themselves. Therefore, F should tend to look like the empirical distribution of the data. Exercise 5 asks you to show why it is better to fit to $i/(m+1)$ rather than i/m.

A similar fit is obtained by using the inverse cdf. Let $Q(u;\theta) = F^{-1}(u;\theta)$, the inverse cdf. If $E[Q(U_{(i)};\theta)]$ is easy to evaluate, then an alternative to (6.10) is to fit directly to the order statistics of X:

$$\widehat{\theta}_O = \operatorname{argmin}_\theta \sum_{i=1}^{m} \left(X_{(i)} - E[Q(U_{(i)};\theta)]\right)^2. \tag{6.11}$$

We refer to both $\widehat{\theta}_U$ and $\widehat{\theta}_O$ as *least-squares fits*.

To employ either the method of moments or least squares matching, we need flexible families of distributions. The flexible distributions that have seen significant application in simulation are Johnson's translation system (see, for instance Swain et al. (1988)) and the generalized lambda distribution (GLD; see for instance Karian and Dudewicz (2000)). Johnson's translation system is comprised of four transformations of the normal distribution, and it covers the entire feasible (α_3^2, α_4) plane; the GLD is a single function that covers a large portion of it. We present the GLD here because it is easy to describe as well as being useful.

The GLD has four parameters $\theta = (\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ and is most easily represented via its inverse cdf

$$Q(u;\theta) = \lambda_1 + \frac{u^{\lambda_3} + (1-u)^{\lambda_4}}{\lambda_2} \tag{6.12}$$

for $0 \le u \le 1$. Therefore, variate generation is immediate. The parameter λ_1 controls location, λ_2 controls scale, and skewness and kurtosis are determined jointly by λ_3 and λ_4.

There is no physical basis for choosing the GLD; its usefulness comes from its flexibility and the ease of random-variate generation. Unfortunately fitting is not as easy, and this is typical of flexible distributions with four or more parameters. Fitting via the method of moments (Karian & Dudewicz, 2000) and least squares (using formulation (6.11), see Lakhany and Mausser (2000) and references therein) are possible, but both involve a numerical search for parameters since there are no closed-form solutions. In the Appendix we give expressions for the necessary moments and expectations.

Figure 6.2 shows the GLD approximation for the N$(0,1)$ distribution, where the circles are points on the normal pdf, while the solid line is the GLD approximation. The standard normal has moments $\mu_X = 0, \sigma_X^2 = 1, \alpha_3 = 0$, and $\alpha_4 = 3$. Clearly out to three standard deviations from the mean, the approximation is quite good. Figure 6.3 shows the GLD approximation to the exponential distribution with mean 1, which has moments $\mu_X = 1, \sigma_X^2 = 1, \alpha_3 = 2$, and $\alpha_4 = 9$. While it has the correct moments, the GLD approximation has a different shape as x approaches 0, and in fact assigns positive (but very, very small) probability to negative x. Matching moments does not guarantee a particular distribution shape.

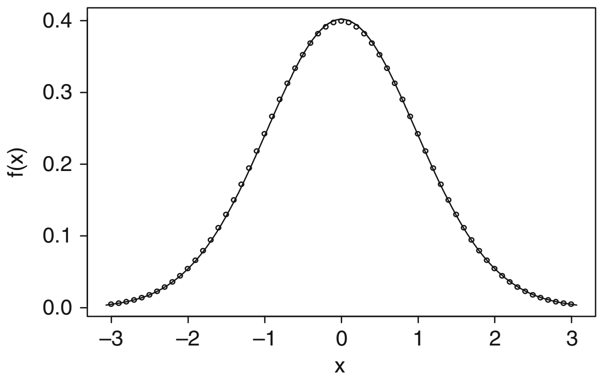

Fig. 6.2 GLD approximation (*solid line*) to points on the standard normal pdf

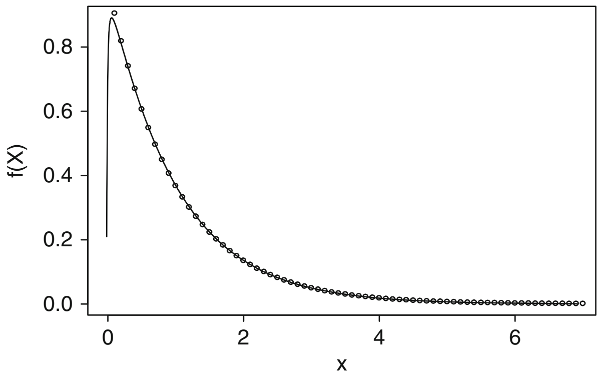

Fig. 6.3 GLD approximation (*solid line*) to points on the exponential pdf with mean 1

The GLD parameters $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ for these examples were obtained from Karian and Dudewicz (2000); for the normal approximation they are $(0, 0.1975, 0.1349, 0.1349)$; while for the exponential approximation they are $(0.006862, -0.0010805, -0.4072 \times 10^{-5}, -0.001076)$.

6.2.4 Empirical Distributions

As in the previous sections, we have a sample of real-world data $X_1, X_2, \ldots, X_m$ that we believe to be i.i.d. observations from some unknown distribution F_X. When there is no strong physical basis for choosing a particular parametric family of distributions, and no parametric distribution provides an adequate fit to the data, then it makes sense to consider using the data themselves. The disadvantages to using the data directly will be apparent shortly.

The empirical cdf (ecdf) is a nonparametric input model defined as

$$\widehat{F}(x) = \frac{1}{m}\sum_{i=1}^{m} I(X_i \le x) \tag{6.13}$$

for all $-\infty < x < \infty$, where $I(\cdot)$ is the indicator function. Let $X_{(1)} \le X_{(2)} \le \cdots \le X_{(m)}$ be the sorted values, and let $\widehat{X}$ be a random variable with distribution $\widehat{F}$. The ecdf places probability mass $1/m$ on each observed value:

$$\begin{aligned}\Pr\left\{\widehat{X} = X_{(i)} \,\middle|\, X_1,\ldots,X_m\right\} &= \Pr\left\{\widehat{X} \le X_{(i)} \,\middle|\, X_1,\ldots,X_m\right\} - \Pr\left\{\widehat{X} < X_{(i)} \,\middle|\, X_1,\ldots,X_m\right\} \\ &= \frac{i}{m} - \frac{(i-1)}{m} = \frac{1}{m}.\end{aligned}$$

This makes variate generation easy via the inverse cdf:

1. Generate $U \sim U(0,1)$
2. Set $i = \lceil mU \rceil$
3. Return $\widehat{X} = X_{(i)}$

The ecdf has several appealing properties. First, as an estimator of the true distribution F_X it is unbiased:

$$\mathrm{E}\left(\widehat{F}(x)\right) = \mathrm{E}\left(\frac{1}{m}\sum_{i=1}^{m} I(X_i \le x)\right) = \frac{m}{m}\mathrm{E}\left(I(X_1 \le x)\right) = F_X(x).$$

Notice that the expectation is with respect to all possible real-world samples from F_X. Further, using the strong law of large numbers it is easy to show that $\widehat{F}(x) \xrightarrow{a.s.} F_X(x)$ pointwise as $m \to \infty$, since $\widehat{F}(x)$ is the average of i.i.d. observations $I(X_i \le x), i = 1,2,\ldots,m$.

A second feature of the edcf is that if we use random variates $\widehat{X}$ in our simulation, then they will have properties that match the sample properties of the observed real-world data; for instance,

$$\mathrm{E}\left(\widehat{X} \,\middle|\, X_1,\ldots,X_m\right) = \sum_{i=1}^{m} X_{(i)}\frac{1}{m} = \bar{X}.$$

Here the expectation is with respect to the edcf $\widehat{F}$, and it shows that the expected value of a random variable with distribution $\widehat{F}$ is the sample mean of the data that created it.

These appealing properties of the ecdf contrast with two undesirable ones: Only the observed values $X_1, X_2, \ldots, X_m$ have positive probability, and only values between $[X_{(1)}, X_{(m)}]$ will be realized. In other words, it is a finite, discrete distribution.

This is particularly troubling if X is known to be continuous valued, or it is expected to have a long tail of extreme but low probability values.

A standard way to obtain a continuous distribution is to linearly interpolate the ecdf between the data points:

$$\tilde{F}(x) = \begin{cases} 0, & x < X_{(1)} \\ \dfrac{i-1}{m-1} + \dfrac{x - X_{(i)}}{(m-1)(X_{(i+1)} - X_{(i)})}, & X_{(i)} \le x < X_{(i+1)} \\ 1, & x \ge X_{(m)}. \end{cases} \tag{6.14}$$

The linearly interpolated ecdf fills in between the observed data; however, the support is still restricted to $[X_{(1)}, X_{(m)}]$. Bratley et al. (1987) describe a linearly interpolated ecdf with an exponential tail added.

Because the linearly interpolated ecdf is piecewise linear, variate generation via inversion is easy:

1. Generate $U \sim U(0,1)$
2. Set $i = \lceil (m-1)U \rceil$
3. Return $\tilde{X} = X_{(i)} + (m-1)(X_{(i+1)} - X_{(i)}) \left(U - \dfrac{i-1}{m-1} \right)$

Unfortunately, $\tilde{F}$ is biased for F_X, although $\tilde{F}(x) \xrightarrow{a.s.} F_X(x)$ as $m \to \infty$. The proof of this is instructive because it shows that various smoothing schemes can be used without losing asymptotic consistency.

Proof. For any fixed x, let

$$\bar{F}(x) = \frac{1}{m-1} \sum_{i=1}^{m} I(X_i \le x) = \frac{m}{m-1} \widehat{F}(x).$$

Then clearly $\bar{F}(x) \xrightarrow{a.s.} F_X(x)$ because $m/(m-1) \to 1$. Notice also that for $X_{(i)} \le x < X_{(i+1)}$ we have

$$0 \le \frac{x - X_{(i)}}{(m-1)(X_{(i+1)} - X_{(i)})} < \frac{1}{m-1}.$$

Therefore,

$$\bar{F}(x) - \frac{1}{m-1} \le \tilde{F}(x) < \bar{F}(x).$$

Since both the lower and upper bounds on $\tilde{F}(x)$ converge a.s. to $F_X(x)$, so does $\tilde{F}(x)$. □

Next consider the random variable $\tilde{X}$. Exercise 10 asks you to show that $\mathrm{E}(\tilde{X} | X_1, \ldots, X_m) \ne \bar{X}$, but the discrepancy decreases with m. Thus, the linearly interpolated ecdf does not perfectly match the sample properties of the data when m is small. Nevertheless, smoothing is typically worthwhile when the input is expected to be continuous valued.

The ecdf is equivalent to resampling the data, with replacement, and the linearly interpolated ecdf fills in the gaps. But what if the number of data values m is very, very large, perhaps multiple millions? Computational issues arise either from trying to embed such a large data set within the simulation, or from referencing it from another source each time a value is needed. However, large m suggests that the ecdf will be a faithful representation of the real input process, which implies that using some version of the ecdf is desirable. The following is a simple way to facilitate this for which many refinements are possible.

Suppose that X is continuous valued. The basic idea is to approximate F_X by linearly interpolating between $k \ll m$ quantiles of F_X, that are estimated precisely using the methods described in Chap. 7.

Let $\widehat{\vartheta}_1 = X_{(1)}$, $\widehat{\vartheta}_k = X_{(m)}$, and

$$\widehat{\vartheta}_i = \widehat{F}_X^{-1}\left(\frac{i-1}{k-1}\right) = X_{(\lceil m\frac{i-1}{k-1}\rceil)},\ i = 2,3,\ldots,k-1.$$

Thus, $\widehat{\vartheta}_i$ is an estimate of the $(i-1)/(k-1)$ quantile of X, $\vartheta_i = F_X^{-1}((i-1)/(k-1))$; see Chap. 7. Assuming a vast amount of data are available, these estimates should be quite precise. This leads to an approximation of the linearly interpolated ecdf algorithm for variate generation:

1. Generate $U \sim U(0,1)$
2. Set $i = \lceil (k-1)U \rceil$
3. Return $\tilde{X} = \widehat{\vartheta}_i + (k-1)(\widehat{\vartheta}_{i+1} - \widehat{\vartheta}_i)\left(U - \frac{i-1}{k-1}\right)$

Notice that obtaining the $\widehat{\vartheta}_i$ is a one-time set-up computation, and k can be quite large (e.g., 1000) without taxing computer memory. We refer to this as the *linearly interpolated quantile method.*

6.2.5 Direct Use of Input Data

Arrivals to the fax center described in Sect. 4.6 were modeled by a nonstationary Poisson arrival process fit to actual arrival data. Fax arrival data are relatively easy to collect since each incoming fax has a time stamp printed on it by the fax machine, and after the fax has been entered we know if it was classified as "simple" or not. Days, months, or even years of such data might be available. This suggests that we could avoid input modeling by driving the simulation with the actual observed arrival data. Functionally, the simulation would read the arrival times and fax types from a file of actual data rather than generating them from an input model. Notice that this is different from using the empirical distribution, which resamples the data rather than using it exactly as it was observed. This distinction is important to understanding the advantages and disadvantages of direct data use:

Advantages: Direct use might capture features missed by an input model, even the empirical distribution. For instance, if our input model treats a process as i.i.d., but there is actually some difficult-to-observe dependence or nonstationary behavior, then this will be reflected in the data but not the input model. In fact, probabilistic input models will seldom be rich enough to capture all of the complexity of a real process.

Disadvantages: Clearly the simulation run length is limited by the amount of data we have; increasing the run length or number of replications is not possible. And the statistical statements we can make are subtle. For instance, in the fax center simulation we could make statements such as "had we staffed at this level during the period when we recorded fax arrivals, our performance would have been. . . ." Statements about long-run performance, however, are more problematic. Consider a system that is affected by rare occurrences in the input (e.g., excessively large insurance claims). However, the system can tolerate isolated occurrences of the rare input value, but not two or more in quick succession. If multiple rare occurrences in close proximity never happened in the observed data, then of course they will never happen in a simulation that directly uses the data; the simulation estimate of the chance of disaster will be 0. However, if we use a probabilistic input model (e.g., modeling insurance claims as being i.i.d. with a fitted distribution F), then over a long enough simulation we will observe multiple rare input values together and recognize the vulnerability of the system.

There is no easy rule for deciding when direct use of the data is appropriate and when fitting an input model is better. When we have large quantities of data that we believe are representative, and we have no reason to believe that the system is dramatically affected by unobserved behavior in the input, then direct data use is reasonable. When we want to make broader statements that go beyond just the input data we saw, then input modeling is essential.

6.2.6 Input Modeling Without Data

Many simulations are undertaken without having relevant input data available. When this happens we try to exploit knowledge about the process to produce reasonable input models, and sensitivity analysis to assess how much they matter.

As discussed in Sect. 6.2, the physical nature of the input process may align well with certain parametric distribution families. This sort of thinking is particularly helpful when there are no data. The difficulty then becomes assigning values to the distribution's parameters. One approach is to obtain subjective estimates from people familiar with the process. However, care must be taken to ask for information in a way that is likely to elicit useful responses. Here is an example.

Suppose the nature of the process suggests use of a normal distribution. Therefore, numerical values for the mean μ and variance σ^2 are required. People are good at providing "typical" values, although a clear distinction between the mean (average) value and the most likely value is not common. Fortunately, for the normal distribution they are the same thing. The variance, however, is very difficult to estimate based on experience alone (without having looked at data). Here are three ways that a value for the variance might be obtained:

Estimate the standard deviation: Unlike the variance, the standard deviation σ is measured in the natural units of the input process, the same units as the mean. Someone familiar with a process might be able to provide the average deviation around the mean, which is an intuitive way to ask for the standard deviation.

Estimate an extreme deviation: If someone familiar with the process can provide an extreme, but still feasible, deviation from the mean, then this could be interpreted as, say, 3σ. Sensitivity analysis could assume multiples of σ that are greater or less than 3.

Relate to a known process: If there is a similar process on which data are available, then one can ask if the other process is more or less variable, and by what percentage. If the process with data has standard deviation S, and the process without data is thought to be 10% less variable, then $\sigma = 0.9S$. Sensitivity analysis could assume values greater or less than 0.9.

The key idea is to translate the parameters of the distribution into terms that are understandable by those familiar with the process.

When there is no strong physical basis for selecting a particular family of distributions, then a distribution like the triangular may be useful. The triangular distribution was not created with a specific physical process in mind; instead it is designed to match three easy-to-specify characteristics: minimum value (a), most likely value (b), and maximum value (c). The pdf of the triangular distribution is

$$f(x) = \begin{cases} \dfrac{2(x-a)}{(b-a)(c-a)}, & a \le x \le b \\ \dfrac{2(c-x)}{(c-b)(c-a)}, & b \le x \le c \\ 0, & \text{elsewhere.} \end{cases} \tag{6.15}$$

Notice that triangular distributions with $a = b$ or $b = c$ are also legitimate distributions where one of the extremes is also the most likely value.

The triangular distribution is typically superior to the uniform distribution on the minimum and maximum values, because there are few real processes for which the extremes (a and c) are just as likely as more central values. One justification for choosing the uniform, however, is that it represents maximum uncertainty on values between a and c.

The uniform and triangular distributions are not the only ones that can be fit from this sort of information. For instance, the beta distribution can also be parameterized by the minimum, maximum, and most likely values. However, there is not a unique beta distribution to represent these characteristics, while the triangular is unambiguously defined. Once there is no longer a physical basis for the distribution choice, then the triangular distribution is as good as any other choice, and it is easy to vary the extremes and the most likely value to check sensitivity.

6.3 Multivariate Input Processes

The previous section described input processes consisting of independent and identically distributed observations; we referred to these as univariate input processes because specification of the marginal distribution is all that is required. This section covers two types of multivariate input processes that are frequently needed in practice: nonstationary arrival processes and random vectors.

Examples of (possibly nonstationary) arrival processes include the arrival of customers to a shopping mall; the arrivals of e-mails to a mail server; and the arrivals of claims to an insurance company. Although we can think of defining the joint distribution of all of the arrival times, it is convenient for discrete-event simulation to describe the *times between arrivals*; this permits the simulation to advance from one arrival to the next by simply scheduling the next arrival.

Examples of random-vector input processes include multiple characteristics of the same customer (e.g., gender, age, income, and occupation); annual sales of related products (e.g., new cars, new car tires, custom floor mats, and GPS devices); and returns on various financial assets (e.g., values of short-term bonds, long-term bonds, and stocks). An input model for a random vector must specify the marginal distribution of each component of the vector and the dependence among them.

There is a vast literature on nonstationary arrival processes that are Poisson, and on multivariate probability distributions that are useful as input models. Good starting points are Leemis (2006) for Poisson arrival processes, and Johnson (1987) and Biller and Ghosh (2006) for random vectors.

Just as was the case with univariate input models, the physical basis of the real process can provide a justification for the choice of multivariate input model. However, models that are obtained in this way tend to be somewhat restrictive. Many well-known multivariate probability distributions have all of their component marginal distributions from the same family. For instance, all marginals of the multivariate normal distribution are normal, making it unsuitable for a situation in which one marginal should be continuous valued, while another is discrete.

For this reason transformation-based approaches for constructing multivariate inputs processes have proven useful in simulation. By “transformation-based approach” we mean that the input process we want is obtained by transforming a more basic input process. To be useful, the base process should have easily controllable characteristics, which the transformation preserves, while the transformation itself allows matching some other desired features of the input process. To be specific, we will obtain nonstationary arrival processes by transforming i.i.d. interarrival times to obtain the desired arrival rate while preserving a measure of variability; and we will obtain random vectors by transforming a vector of dependent $U(0,1)$ random variables to obtain the desired marginal distributions.

The transformation-based approaches we describe provide a general framework for creating arrival processes with desired arrival rate and variability, and random vectors with given marginals and correlations. There is no claim that the physical basis of the real process justifies the transformation. Stated differently, the transformation-based approaches are tools for *matching* characteristics. A convenient feature of this approach is that random-variate generation follows directly: generate the base process, then transform it to obtain inputs. For this reason we will present both input modeling and random-variate generation together in this section.

Time series are a third class of multivariate input process, but they are not covered in this book; a brief introduction with references can be found in Biller and Ghosh (2006). A time-series input process represents a sequence of (possibly vector) random variables that are dependent in sequence, for example the sequence of weekly order quantities that a grocery store chain places for a sports drink.

6.3.1 Nonstationary Arrival Processes

Arrival processes are often primary inputs to computer simulation of real-world systems. The parking lot in Sect. 3.1 had an arrival process representing cars, while the arrivals to the hospital reception in Sect. 3.2 were patients and visitors. A characteristic of these two examples, and many real arrival processes, is that the arrivals are not organized or predictable. In such cases, the usual input modeling paradigm is to characterize the distribution of the time between arrivals, also called the *interarrival time*. The interarrival-time approach does not apply to scheduled arrivals, such as patients to a doctor's office, and may or may not be a good model for arrivals targeting a particular event time, such as fans arriving to a basketball game; in both of these cases the total number of arrivals is fixed (number of appointments or number of tickets sold, respectively) and the arrivals are supposed to occur at or around a specific time. See Exercises 11 and 22.

Renewal processes are the simplest form of random arrivals: In a renewal arrival process, $\tilde{A}_1, \tilde{A}_2, \ldots$ are i.i.d. nonnegative random variables with distribution G that represent the interarrival times of successive entities. The arrivals to the hospital reception were modeled in this way. In addition to the interarrival times, two related stochastic processes are

$$\tilde{S}_n = \begin{cases} 0, & n = 0 \\ \sum_{i=1}^{n} \tilde{A}_i, & n = 1, 2, \ldots \end{cases}$$
$$\tilde{N}(t) = \max\{n \geq 0 : \tilde{S}_n \leq t\}.$$

In words, $\tilde{S}_n$ is the time of the nth arrival, while $\tilde{N}(t)$ is the number of arrivals by time $t \geq 0$. The continuous-time process $\tilde{N}$ is also called the *arrival-counting process*.

Renewal arrival processes are easy to simulate, since an arrival at time $\tilde{S}_n$ triggers scheduling the next arrival to occur at time $\tilde{S}_{n+1} = \tilde{S}_n + \tilde{A}_{n+1}$. If the interarrival times have finite mean and variance, and we let $\tilde{\lambda} = 1/\mathrm{E}(\tilde{A}_i)$, then

$$\frac{\mathrm{E}\left(\tilde{N}(t)\right)}{t} \longrightarrow \tilde{\lambda}$$

as $t \to \infty$, which is why $\tilde{\lambda}$ is interpreted as the arrival rate of the process (see, e.g., Kulkarni (1995)). From here on we will assume that the interarrival times $\tilde{A}_i$ are continuous valued and have a density function, which is appropriate for arrivals that occur in continuous time.[3]

The parking lot did not have a renewal arrival process; instead it had an arrival process with a time-varying arrival rate $\lambda(t) = 1000 + 100\sin(\pi t/12)$. Such arrival processes are called *nonstationary*, and are the focus of this section. We will, however, simulate nonstationary arrival processes by transforming renewal arrival processes in one of two ways: either by stretching (to space arrivals farther apart) or compressing (to force arrivals closer together) the time between renewal arrivals, or by selectively ignoring ("thinning") some renewal arrivals. To be concrete we use the example of the arrival of e-mails to a mail server.

To do this we need the concept of an *equilibrium renewal process*. The only difference between an equilibrium renewal process and the renewal process defined above is that the first interarrival time $\tilde{A}_1$ has distribution G_e, where

$$G_e(t) = \Pr\{\tilde{A}_1 \le t\} = \tilde{\lambda} \int_0^t (1 - G(s))\, \mathrm{d}s. \tag{6.16}$$

Think of this as the distribution of the time until the next arrival if we started observing the renewal arrival process at an arbitrary point in time.[4] If we initialize the renewal process in this way, then it can be shown that

$$\frac{\mathrm{E}\left(\tilde{N}(t)\right)}{t} = \tilde{\lambda} \tag{6.17}$$

for all $t \ge 0$, not just in the limit (Kulkarni, 1995).

Now consider arrival processes with time-varying arrival rates. Let $N(t)$ be the arrival-counting process of a (potentially) nonstationary arrival process. To make the concept of "arrival rate" precise, define

$$\Lambda(t) = \mathrm{E}(N(t))$$

the expected number of arrivals by time t, and the arrival rate by

[3] It is certainly possible to have discrete-time arrival processes when activities are synchronized to a clock, as in some computer networks.

[4] If G is an exponential distribution, then $G_e = G$ due to the memoryless property of the exponential distribution. Exercise 25 asks you to prove this.

$$\lambda(t) = \frac{d}{dt}\Lambda(t)$$

when Λ is differentiable. The function $\Lambda(t)$ is also called the *integrated rate function*. For an equilibrium renewal process, Eq. (6.17) implies that $\Lambda(t) = \tilde{\lambda}t$ and $\lambda(t) = \tilde{\lambda}$; to simulate nonstationary arrival processes, we allow more general $\Lambda(t)$ and $\lambda(t)$.

6.3.1.1 Inverting *Λ(t)*

Suppose that the time-varying behavior that we want is specified through $\Lambda(t)$ (which might be obtained by integrating $\lambda(t)$). Let $\tilde{S}_n$ be an equilibrium renewal arrival process with arrival rate $\tilde{\lambda} = 1$. Define a nonstationary arrival process with arrival times S_n, interarrival times A_n, and arrival-counting process $N(t)$ as follows:

```
1. Set index $n = 1$ and $\tilde{S}_0 = 0$
2. Generate $\tilde{A}_n$
3. Let
    a. $\tilde{S}_n = \tilde{S}_{n-1} + \tilde{A}_n$
    b. $S_n = \Lambda^{-1}(\tilde{S}_n)$
    c. $A_n = S_n - S_{n-1}$
4. $n = n + 1$
5. Go to Step 2
```

Let $N(t) = \max\{n \geq 0 : S_n \leq t\}$. Notice that $\Lambda(t)$ provides a change of time scale: If it is time s for $\tilde{N}$ then it is time $\Lambda^{-1}(s)$ for N; similarly, if it is time t for N, then it is time $\Lambda(t)$ for $\tilde{N}$. Therefore, $N(t) = \tilde{N}(\Lambda(t))$ and

$$\begin{aligned} \mathrm{E}(N(t)) &= \mathrm{E}\left[\mathrm{E}\left(N(t)\,\middle|\tilde{N}(\Lambda(t))\right)\right] \\ &= \mathrm{E}\left[\tilde{N}(\Lambda(t))\right] \\ &= 1 \cdot \Lambda(t) = \Lambda(t) \end{aligned}$$

remembering that $\tilde{N}$ is an equilibrium renewal process with rate 1. *Thus,* Λ^{-1} *transforms the arrival times of an equilibrium renewal process with arrival rate* 1 *into a nonstationary arrival process with rate* $\lambda(t) = d\Lambda(t)/dt$. Inversion provides a very simple way to obtain a nonstationary process with given arrival rate, provided Λ is easily invertible.

Figure 6.4 illustrates the inversion method for an arrival process with $\lambda(t) = 2t$, so that $\Lambda(t) = t^2$. Since the rate is increasing in time, we expect arrivals to be more and more densely packed as t increases. In the figure, the vertical axis represents the arrival times in a rate-1 base process, which is still a random process but with a mean time between arrivals of 1. These times are mapped into arrival times in the nonstationary process by $\Lambda^{-1}(s) = \sqrt{s}$, as shown on the horizontal axis. Notice that as t increases, the arrival rate is clearly increasing as well.

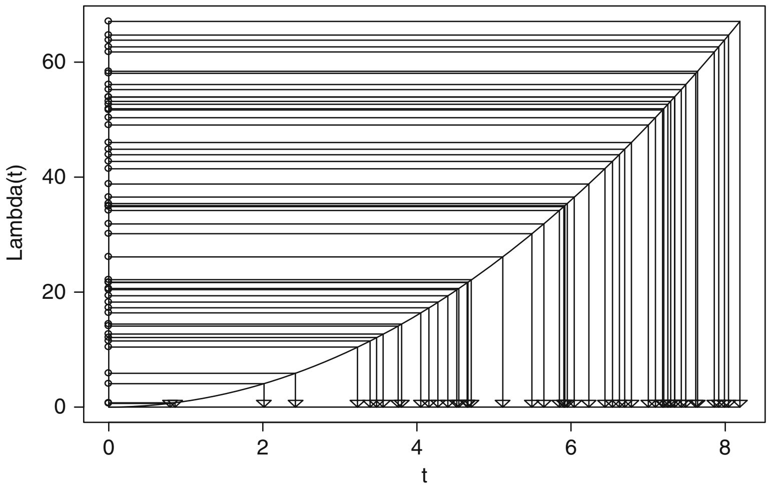

Fig. 6.4 Illustration of the inversion method when $\lambda(t) = 2t$ so that $\Lambda(t) = t^2$ and $\Lambda^{-1}(s) = \sqrt{s}$. The $\circ$'s on the *vertical axis* represent arrival times for the rate-1 base process while the $\bigtriangledown$'s on the *horizontal axis* represent arrival times in the nonstationary process

6.3.1.2 Thinning for $\lambda(t)$

Suppose now that the time-varying behavior that we want is specified through the arrival rate $\lambda(t)$ (which might be obtained by differentiating $\Lambda(t)$). Let $\tilde{S}_n$ be an equilibrium renewal arrival process with arrival rate $\tilde{\lambda} = \max_t \lambda(t)$, which we assume is finite. The idea behind thinning is to generate a stationary process of potential arrivals at the maximum rate, then randomly delete or "thin" some of the potential arrivals to form the actual arrival process. The probability that a potential arrival at time t is thinned is $1 - \lambda(t)/\tilde{\lambda}$. The algorithm is as follows:

```
1. Set indices $n = 1$ and $k = 1$ and $\tilde{S}_0 = 0$
2. Generate $\tilde{A}_n$ and let $\tilde{S}_n = \tilde{S}_{n-1} + \tilde{A}_n$
3. Generate $U \sim U(0,1)$
4. If $U \le \lambda(\tilde{S}_n)/\tilde{\lambda}$ then
   a. $S_k = \tilde{S}_n$
   b. $A_k = S_k - S_{k-1}$
   c. $k = k + 1$
   Endif
5. $n = n + 1$
6. Go to Step 2
```

If we again let $N(t) = \max\{n \ge 0 : S_n \le t\}$, then Gerhardt and Nelson (2009) showed that $\mathrm{E}(N(t)) = \Lambda(t) = \int_0^t \lambda(s)\,\mathrm{d}s$. *Thus, thinning $\tilde{S}_n$ transforms the arrival times of an equilibrium renewal process with arrival rate $\tilde{\lambda} = \max_t \lambda(t)$ into a nonstationary arrival process with rate $\lambda(t) = d\Lambda(t)/dt$.* While inversion and thinning

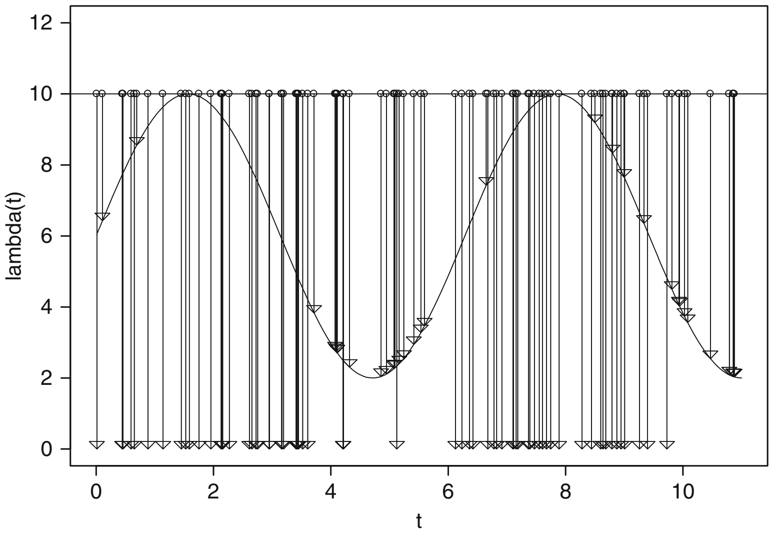

Fig. 6.5 Illustration of the thinning method when $\lambda(t) = 6 + 4\sin(t)$. The ∘'s represent arrival times for the rate-10 base process while the $\bigtriangledown$'s on the *horizontal axis* represent arrival times in the nonstationary process

both lead to arrival processes with the same time-varying arrival rate, they do not give processes that are probabilistically the same, in general (although they do for Poisson arrival processes; see Sect. 6.3.1.4). In other words, inversion and thinning are transformations that both yield the desired arrival rate, but other aspects of the arrivals, such as the variance of the number of arrivals by time t, will be different.

Thinning was the approach used for the parking lot example in Sect. 3.1. Thinning has the distinct advantage of being applicable for any bounded arrival rate $\lambda(t)$, no matter how complex, while inverting $\Lambda(t)$ can be computationally difficult. However, thinning can also be slow if $\tilde{\lambda}$ is greater than much of $\lambda(t)$ because many potential arrivals are generated, but most are thinned. Figure 6.5 illustrates the approach.

6.3.1.3 Estimating $\Lambda(t)$ or $\lambda(t)$ from Arrival Data

We next turn to the topic of estimating $\Lambda(t)$ or $\lambda(t)$.

Suppose that we can observe k independent realizations of an arrival process. Specifically, we observe T_{ij}, the time of the ith arrival on the jth realization for $j = 1, 2, \ldots, k$; and we have reason to believe that realizations are independent and identically distributed. For instance, to model the arrival of e-mail messages to `mail.iems.northwestern.edu` on Mondays, we might observe $k = 10$ Mondays from 4 a.m. (time 0) to 10 p.m. (time T). For input modeling, we want to approximate the arrivals by an arrival-counting process $N(t)$ that has $\mathrm{E}(N(t)) = \Lambda(t)$ over $0 \le t \le T$.

Let $C_j(t)$ be the cumulative number of arrivals by time t on the jth realization; therefore the full data set is

$$\{T_{ij}; i = 1, 2, \ldots, C_j(T)\},\ j = 1, 2, \ldots, k. \tag{6.18}$$

The natural estimator of $\Lambda(t)$ is

$$\bar{\Lambda}(t) = \frac{1}{k} \sum_{j=1}^{k} C_j(t) \tag{6.19}$$

the average number of arrivals by time t over k realizations. Exercise 13 asks you to prove that $\bar{\Lambda}(t) \xrightarrow{a.s.} \Lambda(t)$ as $k \to \infty$, and $\mathrm{E}\left(\bar{\Lambda}(t)\right) = \Lambda(t)$ for any fixed $t \in [0, T]$.

However, $\bar{\Lambda}(t)$ is not a very satisfying estimator. Let $C = \sum_{j=1}^{k} C_j(T)$ be the total number of observed arrivals, and let $T_{(1)} \le T_{(2)} \le \cdots \le T_{(C)}$ be *all* of the arrival times in (6.18) sorted from smallest to largest. Notice that $\bar{\Lambda}(t)$ is a piecewise-constant function that jumps $1/k$ at each arrival time $T_{(i)}$. As a result, if we apply the inversion method to generate arrivals, then only the arrival times $T_{(i)}, i = 1, 2, \ldots, C$ can be generated and nothing in between.

A solution is to linearly interpolate between observed arrival times (Leemis, 1991). Let $T_{(0)} = 0$ and $T_{(C+1)} = T$. Then define

$$\widehat{\Lambda}(t) = \left(\frac{C}{C+1}\right) \left\{ \frac{i}{k} + \frac{1}{k} \left(\frac{t - T_{(i)}}{T_{(i+1)} - T_{(i)}} \right) \right\} \tag{6.20}$$

when $T_{(i)} < t \le T_{(i+1)}$ for $i = 0, 1, \ldots, C$. The factor $C/(C+1)$ is needed because there are $C + 1$ gaps when we include $T_{(0)}$ and $T_{(C+1)}$. Much like the linearly interpolated ecdf, Leemis (1991) showed that $\widehat{\Lambda}(t) \xrightarrow{a.s.} \Lambda(t)$ as $k \to \infty$, so $\widehat{\Lambda}$ fills in the gaps while still giving a consistent estimator of Λ.

To illustrate the two estimators with a (very) small example, suppose we observed an arrival process for $k = 2$ observation periods, each of $T = 5$ h. On the first observation period arrivals occurred at times $T_{11} = 1.2$ and $T_{21} = 4.1$ h, so $C_1(T) = 2$ arrivals. On the second observation period, we observed only $C_2(T) = 1$ arrival at time $T_{12} = 2.4$ h. Therefore, the sorted arrival times are $T_{(1)} = 1.2, T_{(2)} = 2.4$ and $T_{(3)} = 4.1$, and $C = C_1(T) + C_2(T) = 3$.

The natural estimator $\bar{\Lambda}(t)$ jumps $1/k = 1/2$ at each arrival time $T_{(1)} = 1.2, T_{(2)} = 2.4$ and $T_{(3)} = 4.1$ and is constant in between; it is shown as the solid curve in Fig. 6.6. The interpolated estimator $\widehat{\Lambda}(t)$ increases by $(C/(C+1))(1/k) = 3/8$ at each arrival time, and linearly interpolates in between; it is shown as the dashed curve in the figure. Figure 6.7 is a more realistic integrated rate function: The arrival of emergency phone calls during a 24-h period observed over three Mondays.

The algorithm below generates arrival times S_n and interarrival times A_n for a process with integrated rate function $\widehat{\Lambda}$ by inversion. Remember that $\tilde{S}_n$ and $\tilde{A}_n$ are the arrival times and interarrival times, respectively, of an equilibrium renewal process with arrival rate 1. The algorithm is based on Leemis (1991).

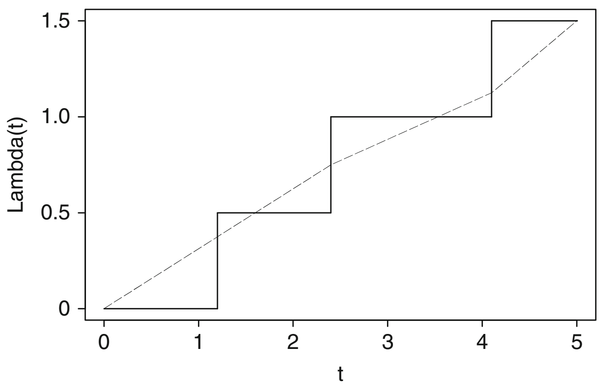

Fig. 6.6 Illustration of the estimators $\bar{\Lambda}$ (*solid*) and $\widehat{\Lambda}$ (*dashed*)

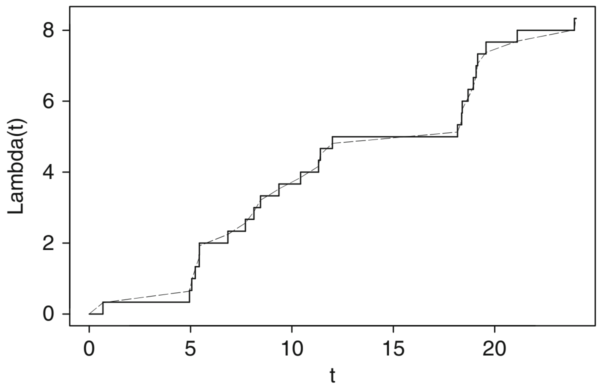

Fig. 6.7 Estimation of $\Lambda(t)$, the expected number of emergency phone calls by time t with $\bar{\Lambda}$ (*solid*) and $\widehat{\Lambda}$ (*dashed*)

```
1. Set $n = 1$ and $S_0 = 0$
2. Generate $\tilde{S}_1 \sim G_e$
3. While $\tilde{S}_n \le C/k$ do
    a. $m = \left\lfloor \left(\frac{C+1}{C}\right) k\tilde{S}_n \right\rfloor$
    b. $S_n = T_{(m)} + \left(T_{(m+1)} - T_{(m)}\right)\left(\left(\frac{C+1}{C}\right) k\tilde{S}_n - m\right)$
    c. $A_n = S_n - S_{n-1}$
    d. $n = n + 1$
    e. Generate $\tilde{A}_n \sim G$
    f. $\tilde{S}_n = \tilde{S}_{n-1} + \tilde{A}_n$
   Loop
```

Notice that the condition $\tilde{S}_n \le C/k$ is needed because $\tilde{S}_n = C/k$ maps into an arrival at time T, which is the end of the observation interval.

Next consider estimating the arrival-rate function $\lambda(t)$ directly. A standard method is to assume that $\lambda(t)$ is piecewise constant over intervals of length $\delta > 0$, for δ small enough. Then $\widehat{\lambda}(t)$ is a piecewise-constant rate function obtained as the average number of arrivals observed in nonoverlapping intervals of size δ.

To be specific, assume T/δ is integer. Then

$$\widehat{\lambda}(t) = \frac{1}{k\delta}\sum_{j=1}^{k}\left[C_j(\ell(t+\delta)) - C_j(\ell(t))\right], \tag{6.21}$$

where $\ell(t) = \lfloor t/\delta \rfloor \delta$ is the beginning of the interval in which time t falls. This is a rather complicated way to express a simple idea: To estimate the arrival rate between, say, times $i\delta < t \le (i+1)\delta$, compute the average number of arrivals that occurred during this interval across the k realizations, then divide by δ to make it a rate. The resulting $\widehat{\lambda}(t)$ can be incorporated into a thinning algorithm, or integrated and used with inversion.

To illustrate, consider again the small example where we observed an arrival process for $k = 2$ observation periods, each of $T = 5$ h. On the first observation period arrivals occurred at times $T_{11} = 1.2$ and $T_{21} = 4.1$ h, while on the second observation period we observed only 1 arrival at time $T_{12} = 2.4$ h. If we set $\delta = 2.5$ h, then there were in total two arrivals between $t = 0 \times \delta = 0$ and $t = 1 \times \delta = 2.5$, and only one arrival between times $t = 2.5$ and $t = 2\delta = 5$. Therefore, the estimated arrival rate is

$$\lambda(t) = \begin{cases} \dfrac{1}{k\delta}2 = \dfrac{2}{5}, \ 0 < t \le \delta = 2.5 \\[2ex] \dfrac{1}{k\delta}1 = \dfrac{1}{5}, \ 2.5 < t \le 2\delta = 5. \end{cases}$$

Selecting δ is clearly difficult: too small and there may be intervals with few or no observed arrivals; too large and the nonstationary behavior may be masked. Henderson (2003) showed a consistency property of this estimator as $k \to \infty$ provided $\delta \to 0$.

For example, Fig. 6.8 shows $\widehat{\lambda}(t)$ constructed from the data on arrivals of emergency phone calls during a 24-h period over three Mondays with $\delta = 1$ h. Notice that the sparseness of the data results in many hours of the day having estimated arrival rate 0; while the arrival rate of emergency calls may be quite small, it is unlikely to be 0 over the long run. However, if we make δ much larger, then the time-of-day effect will be lost; the linearly interpolated estimator $\widehat{\Lambda}(t)$ avoids this problem. To estimate the arrival rate directly it is important to have sufficient data (observation periods).

In some situations it is more convenient to collect arrival counts rather than arrival times. For instance, counts of customer arrivals might be recorded every 30 min. This makes it natural to use the piecewise-constant arrival-rate function (6.21) with

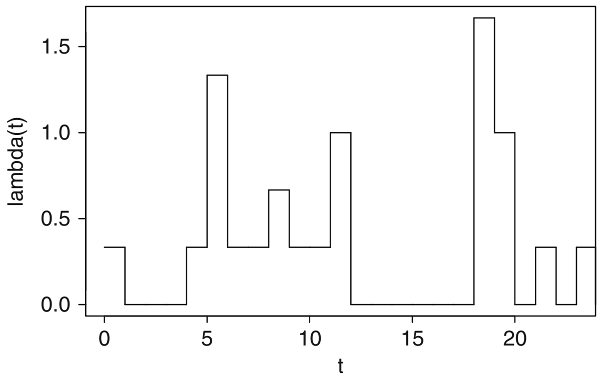

Fig. 6.8 Estimation of $\lambda(t)$, the arrival rate of emergency phone calls at time t, by $\widehat{\lambda}$

$\delta = 30$, the resolution used to collect the data. All else being equal, we recommend using $\widehat{\Lambda}(t)$ when actual arrival times are available, and $\widehat{\lambda}(t)$ when all we have are counts.

Fitting $\Lambda(t)$ or $\lambda(t)$ to data is a very active research area with new and practical methods created regularly; nonparametric, semi-parametric, and fully parametric estimators of $\Lambda(t)$ or $\lambda(t)$ are available. Some recent references with good pointers to the broader literature are Chen and Schmeiser (2019), Morgan et al. (2019), and Zheng and Glynn (2017).

6.3.1.4 Poisson or Not Poisson?

If the interarrival times of the renewal base process are exponential with rate $\tilde{\lambda}$, then the base process is Poisson, and either inversion or thinning leads to probabilistically identical *nonstationary Poisson processes (NSPP)*. The stationary and nonstationary Poisson processes are the most widely used in simulation practice—in fact, they are the default arrival process in many simulation languages—because they are often good representations of arrivals from a large population of potential arrival entities, each making independent decisions about whether or when to arrive (see Sect. 6.2.1.1). They are also very convenient processes to simulate because $G_e(t) = G(t) = 1 - \mathrm{e}^{-\tilde{\lambda}t}, t \geq 0$, so that interarrival times of the base process are easily generated by the inverse cdf method.

How can we recognize that an arrival process is not well-represented as a NSPP, and how can we model the arrival process when it is not? Consider a NSPP with integrated rate function $\Lambda(t)$. One of the many known properties of a NSPP is that

$$\frac{\mathrm{Var}(N(t))}{\mathrm{E}(N(t))} = 1 \text{ for all } t \geq 0. \tag{6.22}$$

In words, the ratio of the variance of the number of arrivals to the expected number of arrivals by time t is always 1, no matter what $\Lambda(t)$ or $\lambda(t)$ is chosen. However, some real arrival processes are more variable (e.g., call centers) or less variable (e.g., manufacturing orders) than this, which is one way to recognize and adjust for departures from Poisson.

Gerhardt and Nelson (2009) showed that if the nonstationary arrival process is generated by the inversion method, then

$$\frac{\mathrm{Var}(N(t))}{\mathrm{E}(N(t))} \approx \sigma_A^2 \text{ for large } t, \tag{6.23}$$

where $\sigma_A^2 = \mathrm{Var}\left(\tilde{A}_2\right)$, the variance of the stationary interarrival times of the rate-1 base process. When the base process is exponential, then $\sigma_A^2 = 1$. Thus, the base process provides a way to obtain arrival processes that are more or less variable than a NSPP.

The key idea is that deviation of the ratio $\mathrm{Var}(N(t))/\mathrm{E}(N(t))$ *from* 1 *is a measure of deviation from being a Poisson process, but the deviation can be matched by using an appropriate base process.*

Assuming we have arrival data, let

$$V(t) = \frac{1}{k-1}\sum_{j=1}^{k}\left(C_j(t) - \bar{\Lambda}(t)\right)^2$$

be the estimated variance of the number of arrivals by time t. If we select a collection of times $t_1 < t_2 < \cdots < t_m$, then an estimator of σ_A^2 is

$$\widehat{\sigma}_A^2 = \frac{1}{m}\sum_{i=1}^{m}\frac{V(t_i)}{\bar{\Lambda}(t_i)}.$$

When this value is significantly different from 1, then it indicates that a NSPP is not appropriate; it also provides an estimate of what the variance of the equilibrium renewal base process should be. In the exercises we suggest some base processes that have controllable variances and for which generating values from G and G_e is not difficult.

Estimating $\widehat{\sigma}_A^2$ from data is illustrated in Table 6.1. Notice that the hourly count data must first be transformed into cumulative counts before computing $\bar{\Lambda}(t)$, $V(t)$, and their ratio $V(t)/\bar{\Lambda}(t)$. The average of these ratios is $\widehat{\sigma}_A^2 = 0.59$, indicating an arrival process that is less variable than Poisson. Therefore, to generate arrivals by inversion we want a renewal base process that is correspondingly less variable than Poisson.

At present there is no easy-to-use relationship between the variance of the base process and the arrival process for thinning, so we only recommend thinning for generating a NSPP.

Notice that the method described in this section for assessing whether a NSPP is appropriate exploited the existence of $k > 1$ arrival-process sample paths. What if there is only one? Nelson and Leemis (2020) show that it is not possible to

Table 6.1 Illustration of estimating $\widehat{\sigma}_A^2$ given hourly arrival count data from $k = 3$ realizations. The # column is the hourly count and $C_j(t)$ is the cumulative count

t	#	$C_1(t)$	#	$C_2(t)$	#	$C_3(t)$	$\bar{\Lambda}(t)$	$V(t)$	$V(t)/\bar{\Lambda}(t)$
8 a.m.–9 a.m.	1	1	3	3	1	1	1.67	1.33	0.80
9 a.m.–10 a.m.	5	6	7	10	4	5	7.00	7.00	1.00
10 a.m.–11 a.m.	14	20	8	18	11	16	18.00	4.00	0.22
11 a.m.–12 p.m.	19	39	15	33	17	33	35.00	12.00	0.34
12 p.m.–1 p.m.	22	61	20	53	18	51	55.00	28.00	0.51
1 p.m.–2 p.m.	9	70	11	64	8	59	64.33	30.33	0.47
2 p.m.–3 p.m.	4	74	1	65	2	61	66.67	44.33	0.67
3 p.m.–4 p.m.	2	76	3	68	1	62	68.67	49.33	0.72
$\widehat{\sigma}_A^2$									0.59

test whether a NSPP is supported by the data from a single sample path without additional stronger assumptions, such as knowing $\lambda(t)$ or $\Lambda(t)$. However, if process physics supports the use of a NSPP, then it is possible to estimate $\Lambda(t)$ from a single path using the interpolation method described in Sect. 6.3.1.3.

6.3.2 Random Vectors

Suppose that the simulation requires generation of a random vector $(X_1, X_2, \ldots, X_k)$, where the ith component has cdf $F_i = F_{X_i}$. Let $(U_1, U_2, \ldots, U_k)$ be a vector of $U(0,1)$ random variables that may or may not be dependent. Then it follows from the development of the inverse cdf method in Sect. 2.2 that the transformation

$$\begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{pmatrix} = \begin{pmatrix} F_1^{-1}(U_1) \\ F_2^{-1}(U_2) \\ \vdots \\ F_k^{-1}(U_k) \end{pmatrix}$$

produces a random vector with the correct marginal distributions, and it seems plausible that the vector $(X_1, X_2, \ldots, X_k)$ will inherit some of the dependence of $(U_1, U_2, \ldots, U_k)$. Therefore, the key to this method is finding a dependent vector $(U_1, U_2, \ldots, U_k)$ that produces the desired dependence among $(X_1, X_2, \ldots, X_k)$. Here we will measure dependence by the correlation matrix $\mathsf{R} = (\rho_{ij})$ where $\rho_{ij} = \mathrm{Corr}(X_i, X_j)$.

While there are a number of ways to construct a base vector $(U_1, U_2, \ldots, U_k)$, we will describe the normal-to-anything (NORTA) method:

$$\begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_k \end{pmatrix} = \begin{pmatrix} F_1^{-1}(U_1) \\ F_2^{-1}(U_2) \\ \vdots \\ F_k^{-1}(U_k) \end{pmatrix} = \begin{pmatrix} F_1^{-1}[\Phi(Z_1)] \\ F_2^{-1}[\Phi(Z_2)] \\ \vdots \\ F_k^{-1}[\Phi(Z_k)] \end{pmatrix} \tag{6.24}$$

where $(Z_1, Z_2, \ldots, Z_k)$ has a standard multivariate normal distribution with correlation matrix $\mathbf{R}$, each component having mean 0 and variance 1, and Φ is the cdf of the standard normal distribution. The NORTA method is based on the fact that if Z is standard normal, then $U = \Phi(Z)$ has a $U(0,1)$ distribution; see Exercise 36. There are many methods for constructing joint distributions of uniforms $(U_1, U_2, \ldots, U_k)$; these distributions are known as *copulas*. The normal copula is a good one for controlling the correlations of $(X_1, X_2, \ldots, X_k)$, but other measures of dependence may be relevant in some applications. A comprehensive reference is Biller and Corlu (2012).

To employ the NORTA method, we first choose or fit the cdfs $F_1, F_2, \ldots, F_k$ of the k component random variables, and also estimate the pairwise correlations $\mathsf{R} = (\rho_{ij})$ that we require. Given $F_1, F_2, \ldots, F_k$ and ρ_{ij}, we find a correlation matrix $\mathbf{R} = (r_{ij})$ for $(Z_1, Z_2, \ldots, Z_k)$ that implies the target correlation matrix R for $(X_1, X_2, \ldots, X_k)$ after the NORTA transformation. Variate generation is done by first generating a standard multivariate normal random vector $(Z_1, Z_2, \ldots, Z_k)$ with correlation matrix $\mathbf{R}$ and then applying the transformation (6.24).

The level of complexity involved in implementing the NORTA method is significantly greater than other methods described here. Details are provided in the Appendix to the chapter. To illustrate we use a very simple example chosen to be easy to understand, rather than corresponding to a practical problem.

Suppose that we need bivariate random vectors (X_1, X_2) where X_1 has an exponential distribution with mean 10, X_2 has a discrete uniform distribution on $\{1, 2, \ldots, 10\}$, and we want to match some $\rho_{12} = \mathrm{Corr}(X_1, X_2)$. These might represent a time to do something and a count of the number of things that are done, respectively, so they are naturally dependent. The NORTA transformation is

$$\begin{pmatrix} X_1 \\ X_2 \end{pmatrix} = \begin{pmatrix} F_1^{-1}(U_1) \\ F_2^{-1}(U_2) \end{pmatrix} = \begin{pmatrix} -10\ln(1-U_1) \\ \lceil 10U_2 \rceil \end{pmatrix} = \begin{pmatrix} -10\ln[1-\Phi(Z_1)] \\ \lceil 10\Phi(Z_2) \rceil \end{pmatrix},$$

where (Z_1, Z_2) have a standard bivariate normal distribution with correlation r_{12}. The key to the NORTA method is finding r_{12} that implies (X_1, X_2) have correlation ρ_{12} after the transformation.

Using the method described in the Appendix to the chapter, we solved the correlation-matching problem for several possible values of ρ_{12} as shown below. Notice that the relationship is not linear. Because of the symmetry of the discrete uniform distribution in this problem, we can obtain correlation $-\rho_{12}$ by using base correlation $-r_{12}$; this is not the case for all pairs of marginal distributions.

ρ_{12}	r_{12}
−0.5	−0.578
0.5	0.578
0.7	0.813
0.8	0.950
0.85	0.991

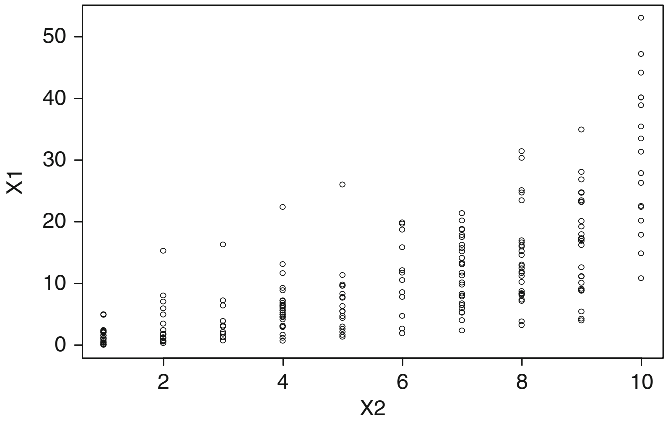

Fig. 6.9 Scatter plot of 200 pairs of bivariate exponential (X_1) and discrete uniform (X_2) variates with correlation 0.7

Figure 6.9 shows a scatter plot of 200 (X_1, X_2) pairs when the correlation is $\rho_{12} = 0.7$. Notice how large values of X_1 tend to occur with large values of X_2.

6.4 Generating Random Variates

The inverse cdf method of random-variate generation was described in Sect. 2.2. For generating univariate inputs this is, in theory, the only method we need. The fact that inversion maps a single uniform U monotonically into a single X via $X = F_X^{-1}(U)$ will be helpful for experiment design and optimization, as discussed in Chap. 9, so inversion is the method of choice.

In practice, however, each inversion may require solving the root-finding problem

$$U = F_X(X) = \int_{-\infty}^{X} f_X(x)\,\mathrm{d}x \tag{6.25}$$

for X, which could be slow if numerical integration is required. In the case of a discrete distribution, a search over a countably infinite number of possible outcomes might be necessary (e.g., the Poisson distribution). Therefore, methods other than inversion are sometimes more practical. Nevertheless, it is worth noting that even when there is no closed-form expression for F_X^{-1}, if (6.25) can be solved efficiently and to sufficient numerical accuracy, then inversion is still preferred.

Random-variate generation is a vast topic with decades of development. An outstanding general reference is Devroye (1986), along with the update in Devroye (2006). The most general method, after inversion, is rejection, which we cover in Sect. 6.4.1. Most other approaches exploit particular properties of the distribution from which variates are needed; we provide a few examples in Sect. 6.4.2.

6.4.1 Rejection Method

Let X be the random variable of interest. The rejection method applies when we can express

$$\Pr\{X \le x\} = \Pr\{V \le x | \mathscr{A}\},$$

where V is another random variable and $\mathscr{A}$ is some "acceptance" event (for this reason the method is sometimes called "acceptance–rejection"). If V is easy to generate, while X is difficult, then this suggests that we can generate X in the following way:

```
1. Generate $V$
2. If $\mathscr{A}$ occurs, then return $X = V$
   Otherwise, reject $V$ and go to Step 1
```

Provided generation of V is fast and, on average, we do not have to reject too many V's until the event $\mathscr{A}$ occurs, then rejection can be a competitive method.

How can we set up such a situation? Suppose that X has a density function f_X (and cdf F_X), and let $m(x)$ be a function that "majorizes" f_X, by which we mean $m(x) \ge f_X(x)$ for all x. Unless they are equal, $m(x)$ will not be a density function (its integral will be greater than 1), but

$$g(x) = \frac{m(x)}{\int_{-\infty}^{\infty} m(y)\,\mathrm{d}y} = \frac{m(x)}{c}$$

will be. The following is a general rejection algorithm:

```
1. Generate $V \sim g$
2. Generate $U \sim U(0,1)$
3. If $U \le f_X(V)/m(V)$ then return $X = V$
   Otherwise go to Step 1
```

Here the acceptance event is $\mathscr{A} = \{U \le f_X(V)/m(V)\}$, and we need to show that

$$\Pr\{V \le x | \mathscr{A}\} = F_X(x).$$

By definition

$$\Pr\{V \le x | \mathscr{A}\} = \frac{\Pr\{V \le x, \mathscr{A}\}}{\Pr\{\mathscr{A}\}}.$$

But

$$\begin{aligned}
\Pr\{\mathscr{A}\} &= \Pr\{U \le f_X(V)/m(V)\} \\
&= \int_{-\infty}^{\infty} \Pr\{U \le f_X(y)/m(y) | V = y\} g(y)\,\mathrm{d}y \\
&= \int_{-\infty}^{\infty} \frac{f_X(y)}{m(y)} \cdot \frac{m(y)}{c}\,\mathrm{d}y \\
&= \frac{1}{c}\int_{-\infty}^{\infty} f_X(y)\,\mathrm{d}y = \frac{1}{c}. \qquad (6.26)
\end{aligned}$$

A similar argument gives the numerator:

$$\begin{aligned}\Pr\{V \leq x, U \leq f_X(V)/m(V)\} &= \int_{-\infty}^{\infty} \Pr\{y \leq x, U \leq f_X(y)/m(y) | V = y\} g(y)\,\mathrm{d}y \\ &= \int_{-\infty}^{x} \Pr\{U \leq f_X(y)/m(y)\} g(y)\,\mathrm{d}y \\ &= \frac{1}{c}\int_{-\infty}^{x} f_X(y)\,\mathrm{d}y = \frac{1}{c} F_X(x). \end{aligned} \tag{6.27}$$

Combining (6.27) and (6.26) proves the result.

Notice that (6.26) is the probability that we accept V on any trial; since the trials are independent, they have a geometric distribution and the expected number of trials to generate one X is c. Thus, the closer c is to 1 the more efficient the algorithm is. Also notice that the rejection method requires a random number of $U(0,1)$'s to generate each X.

As an example, consider the beta distribution with pdf

$$f_X(x) = \frac{x^{\alpha_1 - 1}(1-x)^{\alpha_2 - 1}}{\mathrm{B}(\alpha_1, \alpha_2)} \tag{6.28}$$

for $0 \leq x \leq 1$, where $\alpha_1, \alpha_2 > 0$ are shape parameters, and

$$\mathrm{B}(\alpha_1, \alpha_2) = \int_0^1 u^{\alpha_1 - 1}(1-u)^{\alpha_2 - 1}\,\mathrm{d}u.$$

The beta distribution does not have a closed-form inverse cdf.

When $\alpha_1 > 1$ and $\alpha_2 > 1$ the mode of the beta distribution occurs at $x^\star = (\alpha_1 - 1)/(\alpha_1 + \alpha_2 - 2)$. Thus, the highest point on the density function is $f^\star = f_X(x^\star)$ and the constant function

$$m(x) = f^\star,\ 0 \leq x \leq 1$$

majorizes f_X. This implies that $g(x) = 1$, for $0 \leq x \leq 1$, the uniform distribution on $(0,1)$. This gives the following rejection algorithm for generating beta random variates when $\alpha_1 > 1$ and $\alpha_2 > 1$:

```
0. Compute $f^\star = f_X\left(\frac{\alpha_1 - 1}{\alpha_1 + \alpha_2 - 2}\right)$
1. Generate $V \sim g = U(0,1)$
2. Generate $U \sim U(0,1)$
3. If $U \leq f_X(V)/m(V) = [V^{\alpha_1 - 1}(1-V)^{\alpha_2 - 1}/\mathrm{B}(\alpha_1, \alpha_2)]/f^\star$ then return
   $X = V$
   Otherwise go to Step 1
```

The efficiency of this rejection algorithm depends on the particular values of α_1 and α_2. For instance, when $\alpha_1 = 4$ and $\alpha_2 = 3$, then $c = 2.0736$ meaning the expected number of trials is a little greater than 2 to generate each beta variate. By modern standards this is not a very efficient rejection algorithm. This particular example is illustrated in Fig. 6.10.

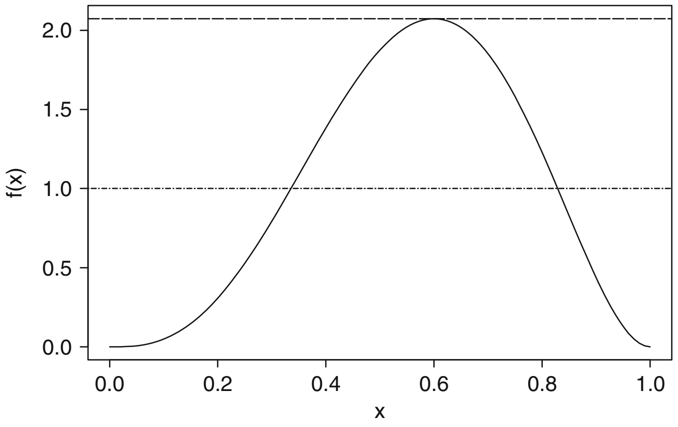

Fig. 6.10 Rejection method for a beta distribution (*solid line*), with majorizing function $m(x)$ (*dashed line*) and corresponding density $g(x)$ (*dot-dashed line*)

State-of-the-art algorithms use a number of tricks to increase efficiency. One is to express the density f_X as a probabilistic mixture of pieces so that each piece can be majorized very closely.

6.4.2 Particular Properties

By "particular properties" we mean relationships among distributions that facilitate variate generation. For instance, if $X_1 \sim \text{gamma}(\alpha_1, \beta)$, independent of $X_2 \sim \text{gamma}(\alpha_2, \beta)$, then $X = X_1/(X_1 + X_2)$ has a beta(α_1, α_2) distribution. Thus, if one has a gamma random-variate generation algorithm, then it can be used to generate beta variates.

Relationships among the standard normal distribution, the normal distribution, and the lognormal distribution have already been mentioned: If $Z \sim \text{N}(0,1)$, then $X = \mu + \sigma Z$ has a $\text{N}(\mu, \sigma^2)$ distribution. Further, $Y = \exp(X)$ has a lognormal distribution. A good reference for such relationships is Leemis and McQueston (2008).

Although particular properties can often supply competitive random-variate generation algorithms, they should typically be used only if they provide exact methods. Recall that what we mean by "exact" is that if we had infinite-precision computer arithmetic, and a source of truly $U(0,1)$ random numbers, then $\Pr\{X \le x\} = F_X(x)$. There are particular properties that only lead to approximate methods. For instance, if X has a Poisson distribution with mean λ and λ is large, then $(X - \lambda)/\sqrt{\lambda}$ is approximately standard normal. Thus, if we had a method for generating $N(\lambda, \lambda)$ random variates, then we could round them to obtain approximately Poisson distributed variates. However, there is no reason to settle for this approximation since exact, uniformly fast methods exist (e.g., Hörmann (1993)).

The most infamous example of an approximate method is generating a $\mathrm{N}(0,1)$ random variate by setting

$$Z = \sum_{i=1}^{12} U_i - 6, \tag{6.29}$$

where $U_1, U_2, \ldots, U_{12}$ are i.i.d. $U(0,1)$. The justification for normality is the central limit theorem, and the convenient choice of 12 gives variance 1. In Exercise 24 you are asked to explore the faults of this method.

6.5 Generating Pseudorandom Numbers

The driving source of randomness in stochastic simulations is, in theory, i.i.d. samples from the $U(0,1)$ distribution; but in practice they are a deterministic sequence of pseudorandom numbers that appear to be random. As described in Sect. 2.3, one way to visualize the pseudorandom numbers is as a large, ordered list

$$u_1, u_2, u_3, \ldots, u_i, u_{i+1}, \ldots, u_{P-1}, u_P, u_1, u_2, u_3, \ldots$$

where P is the period. Here we use lowercase u to denote pseudorandom numbers to emphasize that they are deterministic and not actually random.

Rather than storing an actual list, pseudorandom numbers are invariably generated by a recursive algorithm called a *pseudorandom-number generator*. In this section, we present enough background about pseudorandom-number generators to use them effectively. It is worth stating that inventing algorithms to generate a sequence that appears to be i.i.d. $U(0,1)$ is quite difficult. Basic properties, such as having the right mean (1/2) and variance (1/12), are relatively easy to achieve, but apparent independence is not. For instance, given any dimension d, the values $(u_{i+1}, u_{i+2}, \ldots, u_{i+d}), i = 1,2,\ldots$ should appear to be uniformly distributed in the d-dimensional unit hypercube $(0,1)^d$. This property is hard to achieve as d increases, and poor generators can give results that do not look random in high dimensions. Creating pseudorandom-number generators that have a long period P and good statistical properties involves number theory, computer science, and clever testing. A very accessible reference is L'Ecuyer (2006), to which the reader should refer for technical details not covered here.

The pseudorandom-number generators we consider produce a sequence of integer values that are scaled to be between 0 and 1. The generator in PythonSim starts with an integer seed $z_0 \in \{1, 2, \ldots, 2^{31}-1\}$, and then generates pseudorandom numbers via the recursion

$$\begin{aligned} z_i &= 630{,}360{,}016 z_{i-1} \bmod 2^{31}-1 \\ u_i &= \frac{z_i}{2^{31}-1}. \end{aligned} \tag{6.30}$$

This is a Python implementation of the algorithm described in Marse and Roberts (1983) and Law (2007). Recall that "mod m" means to retain the remainder after dividing by m; for instance, 7 mod 3 = 1. This generator is one instance of a class of generators of the form

$$\begin{aligned} z_i &= az_{i-1} \bmod m \\ u_i &= \frac{z_i}{m} \end{aligned} \tag{6.31}$$

which are called (linear) multiplicative congruential generators (MCGs). MCGs have been in widespread use for many years. Here are some important observations about MCGs:

1. The only possible values a MCG can return are $\{0, 1/m, 2/m, \ldots, (m-1)/m\}$. Clearly we do not want it to return 0 since it will produce nothing but zeroes from then on.[5] Thus, the maximal period of a MCG is $P = m - 1$, which the generator (6.30) achieves.
2. We want m to be large so that the interval between 0 and 1 is densely filled; thus, $m = 2^{31} - 1$ is a natural choice because it is the largest integer representable on 32-bit computers. However, to appear random requires more than just a long period; there are 534 million values of a that achieve full period for $m = 2^{31} - 1$, but only a fraction of them generate sequences with good statistical properties.
3. The computation $z_i = az_{i-1}$ mod m must be done with care. For example, the multiplication $630{,}360{,}016 z_{i-1}$ will frequently yield a value greater than $2^{31} - 1$, leading to overflow if we try to do integer arithmetic.
4. Exercise 45 asks you to show that $z_i = a^i z_0$ mod m for a MCG. Therefore, it is possible to skip ahead in the sequence since

$$z_i = a^i z_0 \bmod m = (a^i \bmod m) z_0 \bmod m.$$

   Thus, a single generator with a long period can act as several virtual generators, each one starting with their own unique "seed" z_0 spaced far apart. The PythonSim implementation of the generator (6.30) has 100 virtual generators with seeds spaced 100,000 pseudorandom numbers apart; these are accessed via the random-number streams 1–100. For instance, stream 1 corresponds to $z_0 = 1{,}973{,}272{,}912$ while stream 2 has $z_0 = 281{,}629{,}770$.

The pseudorandom-number generator (6.30) is adequate for learning about stochastic simulation, but with a period of only about two billion it is insufficient for many problems. Consider this simple and very realistic example: We want to estimate the reliability of a system for which failure needs to be very unlikely because human life is involved. Suppose each replication of the simulation requires 100 random numbers, and we would like to observe at least 30 failures. If the actual probability of failure is one in a million, then we would expect to

[5] Typically we do not want pseudorandom-number generators to return either 0 or 1 because these values correspond to $\pm\infty$ for some distributions from which we want to generate random variates. For instance, the inverse cdf of the exponential distribution with mean 1 is $F^{-1}(u) = -\ln(1-u)$.

need $100 \times 30 \times 1{,}000{,}000 = 3{,}000{,}000{,}000$ random numbers, many more than the pseudorandom-number generator (6.30) can provide before repeating.

Unfortunately, even if the period P of a generator is sufficient for the applications we have in mind problems can occur. L'Ecuyer and Simard (2001) demonstrated that for many standard generators, such as (6.30), nonrandom behavior of the generator is easily detectable by statistical tests if more than a fraction of the period is actually used (from $P^{1/3}$ to $P^{1/2}$). This implies that we would like to have generators with periods substantially larger than what the application requires. To complete our understanding of modern pseudorandom-number generators, we describe two general approaches that greatly extend the period of MCGs; selecting the specific constants that give good statistical properties is, however, beyond the scope of this book.

6.5.1 Multiple Recursive Generators

A recursive pseudorandom-number generator maps a previous state into a new state, so its period depends on the number of distinct states it can take. The period of a MCG is limited because its state is the most recently generated integer z_{i-1}, which can assume only m values. A multiple recursive generator (MRG) of order K extends the state to include the K most recent values:

$$z_i = (a_1 z_{i-1} + a_2 z_{i-2} + \cdots + a_K z_{i-K}) \bmod m$$

$$u_i = \begin{cases} \dfrac{z_i}{m+1}, & z_i > 0 \\ \dfrac{m}{m+1}, & \text{otherwise.} \end{cases} \tag{6.32}$$

Notice that a value of $z_i = 0$ is not fatal for a MRG, but is mapped into $m/(m+1)$ to avoid generating $u_i = 0$. Since the state of this generator is $(z_{i-K}, z_{i-K+1}, \ldots, z_{i-1})$, and since each element can take m possible values, the maximal period—avoiding the state $(0,0,\ldots,0)$—is $P = m^K - 1$. A necessary (but not sufficient) condition for achieving full period is that a_K and at least one other a_j are not zero. Notice that the "seed" for such a generator consists of K values $(z_0, z_1, \ldots, z_{K-1})$.

6.5.2 Combined Generators

Combining MRGs is a way to further increase the state of the generator. Suppose that we have $j = 1, 2, \ldots, J$ MRGs of the form

$$z_{i,j} = (a_{1,j} z_{i-1,j} + a_{2,j} z_{i-2,j} + \cdots + a_{K,j} z_{i-K,j}) \bmod m_j.$$

That is, $z_{i,j}$ is the ith pseudorandom integer from the jth MRG. Then we form the ith pseudorandom number from

$$z_i = (\delta_1 z_{i,1} + \delta_2 z_{i,2} + \cdots + \delta_J z_{i,J}) \bmod m_1$$
$$u_i = \begin{cases} \dfrac{z_i}{m_1+1}, & z_i > 0 \\ \dfrac{m_1}{m_1+1}, & \text{otherwise} \end{cases}$$

where $\delta_1, \delta_2, \ldots, \delta_J$ are fixed integers. In words, the generator consists of J MRGs of order K executed in parallel, with their outputs combined to form a pseudorandom number.

If each of the component MRGs uses a prime number modulus m_j and has period $P_j = m_j^K - 1$, then the period of the combined generator is at most $P = P_1 P_2 \cdots P_J / 2^{J-1}$. Notice that the initial "seed" for such a generator consists of $J \times K$ values.

A specific instance that has been widely implemented is `MRG32k3a` (L'Ecuyer, 1999), which consists of $J = 2$ MRGs each of order $K = 3$:

$$\begin{aligned} z_{i,1} &= (1{,}403{,}580\, z_{i-2,1} - 810{,}728\, z_{i-3,1}) \bmod 2^{32} - 209 \\ z_{i,2} &= (527{,}612\, z_{i-1,2} - 1{,}370{,}589\, z_{i-3,2}) \bmod 2^{32} - 22{,}853 \\ z_i &= (z_{i,1} - z_{i,2}) \bmod 2^{32} - 209. \end{aligned}$$

The period of this generator is $P \approx 2^{191} \approx 3 \times 10^{57}$, which is enormous: If you could generate two billion pseudorandom numbers per second (the entire period of the PythonSim generator), then it would take longer than the age of the universe to exhaust the period of `MRG32k3a`!

There are reasons why combining MRGs can lead to a generator with good statistical properties as well as long period, but again the specific choices must be made with great care; see L'Ecuyer (2006). The following result shows that if the first component MRG $z_{i,1}$ provides what appear to be uniformly distributed integers, then the combined generator will as well. Thus, combining several MRGs will not destroy the uniformity of the first generator. Of course, this provides no guarantee that successive values appear to be independent.

Theorem 6.2 (L'Ecuyer (1988)). *Suppose that $Z_1, Z_2, \ldots, Z_J$ are independent, integer-valued random variables and that Z_1 has a discrete uniform distribution on $\{0, 1, \ldots, m_1 - 1\}$. Then*

$$Z = \sum_{j=1}^{J} Z_j \bmod m_1$$

also has a discrete uniform distribution on $\{0, 1, \ldots, m_1 - 1\}$.

Proof. Let $W = \sum_{j=2}^{J} Z_j \bmod m_1$, and notice that $Z_1 \bmod m_1 = Z_1$. Then

$$\begin{aligned} Z &= \left(Z_1 \bmod m_1 + \sum_{j=2}^{J} Z_j \bmod m_1 \right) \bmod m_1 \\ &= (Z_1 + W) \bmod m_1. \end{aligned}$$

Then since W only takes values in $\{0, 1, \ldots, m_1 - 1\}$, we have for $z = 0, 1, \ldots, m_1 - 1$ that

$$\begin{aligned} \Pr\{Z = z\} &= \sum_{w=0}^{m_1-1} \Pr\{Z = z | W = w\} \Pr\{W = w\} \\ &= \sum_{w=0}^{m_1-1} \Pr\{(Z_1 + W) \bmod m_1 = z | W = w\} \Pr\{W = w\} \\ &= \sum_{w=0}^{m_1-1} \Pr\{(Z_1 + w) \bmod m_1 = z\} \Pr\{W = w\} \\ &= \sum_{w=0}^{m_1-1} \frac{1}{m_1} \Pr\{W = w\} \qquad (6.33) \\ &= \frac{1}{m_1}, \end{aligned}$$

where (6.33) follows because there is only one value of Z_1 that will make $(Z_1 + w) \bmod m_1 = z$, and Z_1 has a discrete uniform distribution so all values are equally likely. □

6.5.3 Proper Use of Pseudorandom-Number Generators

Pseudorandom-number generators are implemented so that after each call they retain memory of the generator's state. Thus, each call to the generator advances to the next pseudorandom number in the sequence, approximating an independent sample from the $U(0,1)$ distribution. This is why we may treat replications as (pseudo) independent, as long as we do not exhaust the period P of the generator or mistakenly reset the initial seed between replications. Notice that a MCG will generate a distinct integer z_i on each call, while a MRG or combined MRGs can (and will) generate the same z_i value many times before exhausting its period, since different generator states map into the same z_i value.

For a well-tested pseudorandom-number generator such as `MRG32k3a`, there is no advantage to using any particular initial seed. A common misconception is that pseudorandom-number generators include different streams (initial seeds) because some parts of the sequence are better (more random) than others, or that assigning different streams makes different input processes more independent; this is not the case. A good pseudorandom-number generator appears to produce a sequence of i.i.d. $U(0,1)$'s, so it does not matter if they are used in their natural order or if they are obtained from different subsequences.

Of course, surprising events occur in a pseudorandom sequence just as they would in a truly random process. This is why adequate run lengths or numbers of replications are essential to overcome the quirks of randomness, and why statistical error should be measured by a standard error or a confidence interval. These are the goals of good experiment design and analysis, the subjects of Chaps. 7 and 8.

One reason streams exist is to synchronize how the random numbers are used when we compare alternative scenarios via simulation. Roughly speaking, we want each scenario to see the same source of randomness so that observed differences in performance are due to structural differences in the scenarios, not different random numbers. As discussed in Chap. 9, this is facilitated by insuring (as far as possible) that each pseudorandom number is used for the same purpose in the simulation of each alternative scenario. Assigning distinct streams to distinct input processes facilitates this. If only a single scenario is simulated, then streams are irrelevant, unless replications are executed in parallel.

The ability to execute the simulations of different scenarios, and different replications of those scenarios, in parallel introduces another wrinkle into the proper use of pseudorandom-number generators. Consider obtaining n replications of $k \geq 1$ scenarios using $1 \leq p \leq n$ parallel processors. First suppose that $k = 1$. When $p = 1$ the replications are executed one after another, and therefore as long as we do not reinitialize the pseudorandom-number generator between replications, the replications will be (pseudo) i.i.d. Clearly this approach fails when $p > 1$ because it defeats the purpose of parallelization to wait for one replication to complete before starting the next one! Therefore, each replication must be assigned its own stream or seed. When there are $k > 1$ scenarios, then the replications for the each scenario must have completely distinct streams or seeds if we want to simulate them independently, or be coordinated if we desire synchronization as described above.

Conceptually, we would like to break the period of a pseudorandom number generator into an effectively infinite number of streams, with an effectively infinite number of substreams within each stream, all effectively infinitely far apart. This facilitates assigning "streams" to scenarios and "substreams" to replications as we see fit. Fortunately, generators such as `MRG32k3a` have long enough periods to make this possible, with the additional feature that the seeds for new starting streams or substreams can be *computed* as needed, rather than having to be stored; see L'Ecuyer et al. (2002) and Exercise 44. The summary point is that more care is required to use pseudorandom-number generators correctly in parallel simulation.

Finally, it is sometimes argued that to appropriately represent the uncertainty that occurs in reality, it is important to randomly select the initial seed z_0. While this may appear correct on the surface, remember that the goal of a stochastic simulation is not to produce a random result, as in a game of cards or in a lottery, but rather to estimate underlying properties of a system that is subject to uncertainty. Whether the initial seed is fixed or random, the experiment should be conducted so as to overcome the effect of randomness and obtain precise estimates. If the pseudorandom-number generator can be trusted, then the benefits of repeatability and precise comparisons that can be obtained with fixed seeds or streams outweigh the superficial connection to true uncertainty provided by choosing a random seed.

Appendix 1: Properties of the GLD

The results below can be found in Karian and Dudewicz (2000) or Lakhany and Mausser (2000).

The pdf of the GLD is

$$f(x) = \frac{\lambda_2}{\lambda_3 y^{\lambda_3 - 1} + \lambda_4 (1-y)^{\lambda_4 - 1}}$$

at the point $x = Q(y)$, where Q is the inverse cdf. This expression is useful for plotting a fitted GLD distribution. The support of the GLD depends on its parameters (Karian & Dudewicz, 2000):

λ_3	λ_4	Support
$\lambda_3 > 0$	$\lambda_4 > 0$	$[\lambda_1 - 1/\lambda_2, \lambda_1 + 1/\lambda_2]$
$\lambda_3 > 0$	$\lambda_4 = 0$	$[\lambda_1, \lambda_1 + 1/\lambda_2]$
$\lambda_3 = 0$	$\lambda_4 > 0$	$[\lambda_1 - 1/\lambda_2, \lambda_1]$
$\lambda_3 < 0$	$\lambda_4 < 0$	$(-\infty, \infty)$
$\lambda_3 < 0$	$\lambda_4 = 0$	$(-\infty, \lambda_1 + 1/\lambda_2]$
$\lambda_3 = 0$	$\lambda_4 < 0$	$[\lambda_1 - 1/\lambda_2, \infty)$

To fit a GLD via the method of moments, the following are valid provided $\lambda_3 > -1/4$ and $\lambda_4 > -1/4$:

$$\mu = \lambda_1 + \frac{A}{\lambda_2}$$
$$\sigma^2 = \frac{B - A^2}{\lambda_2^2}$$

$$\alpha_3 = \frac{C - 3AB + 2A^3}{\lambda_2^3 \sigma^3}$$
$$\alpha_4 = \frac{D - 4AC + 6A^2 B - 3A^4}{\lambda_2^4 \sigma^4},$$

where

$$\begin{aligned}
A &= \frac{1}{1+\lambda_3} - \frac{1}{1+\lambda_4} \\
B &= \frac{1}{1+2\lambda_3} + \frac{1}{1+2\lambda_4} - 2\mathsf{B}(1+\lambda_3, 1+\lambda_4) \\
C &= \frac{1}{1+3\lambda_3} - \frac{1}{1+3\lambda_4} - 3\mathsf{B}(1+2\lambda_3, 1+\lambda_4) + 3\mathsf{B}(1+\lambda_3, 1+2\lambda_4) \\
D &= \frac{1}{1+4\lambda_3} + \frac{1}{1+4\lambda_4} - 4\mathsf{B}(1+3\lambda_3, 1+\lambda_4) + 6\mathsf{B}(1+2\lambda_3, 1+2\lambda_4) \\
&\quad -4\mathsf{B}(1+\lambda_3, 1+3\lambda_4)
\end{aligned}$$

with

$$\mathsf{B}(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1}\,\mathrm{d}x.$$

Software can be found in Karian and Dudewicz (2000).

To fit via least squares we need $\mathrm{E}[Q(U_{(i)})]$, which can be computed using the following:

$$\begin{aligned}
\mathrm{E}\left[U_{(i)}^{\lambda_3}\right] &= \frac{\Gamma(m+1)\Gamma(i+\lambda_3)}{\Gamma(i)\Gamma(m+\lambda_3+1)} \\
\mathrm{E}\left[(1-U_{(i)})^{\lambda_4}\right] &= \frac{\Gamma(m+1)\Gamma(m-i+\lambda_4+1)}{\Gamma(m-i+1)\Gamma(m+\lambda_4+1)}.
\end{aligned}$$

Appendix 2: Implementation of the NORTA Method

In this appendix, we provide details on one implementation of the NORTA method. We assume that you start with observed input data, fit the NORTA input model, and then generate variates.

Fitting NORTA Components

Suppose we have observed input data that we believe are i.i.d. vectors $\mathbf{X}_h = (X_{1h}, X_{2h}, \ldots, X_{kh})^\top, h = 1, 2, \ldots, m$. Since the random vectors are i.i.d., then so are $X_{i1}, X_{i2}, \ldots, X_{im}$ for any one of the marginal distributions i. Thus, the ith marginal may be fit by using any of the methods described in Sect. 6.2. Denote the fitted marginals as $\widehat{F}_i, i = 1, 2, \ldots, k$.

Let

$$\bar{\mathbf{X}} = \frac{1}{m}\sum_{h=1}^{m}\mathbf{X}_h$$

be the sample mean vector, and

$$\widehat{\mathsf{C}} = \frac{1}{m-1}\sum_{h=1}^{m}(\mathbf{X}_h - \bar{\mathbf{X}})(\mathbf{X}_h - \bar{\mathbf{X}})^\top \tag{6.34}$$

be the sample variance–covariance matrix. The sample correlations are therefore $\widehat{\rho}_{ij} = \widehat{\mathsf{C}}_{ij}/\sqrt{\widehat{\mathsf{C}}_{ii}\widehat{\mathsf{C}}_{jj}}$. Let $\widehat{\mathsf{R}} = (\widehat{\rho}_{ij})$ be the estimated correlation matrix. The fitted marginals $\widehat{F}_i, i = 1,2,\ldots,k$ and correlations $\widehat{\rho}_{ij}$ are the inputs to the NORTA correlation-matching problem; we describe one solution method below.

Under the set-up described above, $\widehat{\mathsf{R}}$ is guaranteed to be positive semi-definite, a requirement to be a legitimate correlation matrix. Unfortunately, if we do not have m complete vectors of observations on all k input processes, then there is no such guarantee. For instance, (X_1, X_2) might have been observed together 100 times, while a separate study observed (X_2, X_3) together 87 times, but (X_1, X_3) might never have been observed jointly so a guess of their correlation will be used. From these data we can estimate ρ_{12}, ρ_{13}, and ρ_{23} separately, but there is no longer any guarantee that $\widehat{\mathsf{R}}$ will be positive semi-definite and some sort of remedial measures will be required. For instance, if $\widehat{\mathsf{R}}$ is not a positive semi-definite correlation matrix, then

$$\widehat{\mathsf{R}}' = \frac{\widehat{\mathsf{R}} - \varepsilon\mathbf{I}}{1-\varepsilon}$$

will be, provided ε is the smallest eigenvalue of $\widehat{\mathsf{R}}$ (which will be negative) and $\mathbf{I}$ is the $k \times k$ identity matrix. There are many other fixes; see, for instance, Ghosh and Henderson (2002).

Remark 6.1. Notice that our approach is to decompose the problem into fitting the marginals and the correlations separately, and then solving the correlation-matching problem. Biller and Nelson (2005) describe a least-squares method for fitting the marginals and correlation structure simultaneously.

Generating Multivariate Normal Vectors

A component of the NORTA method is generating standard multivariate normal random vectors with correlation matrix $\mathbf{R}$. The multivariate normal distribution is completely characterized by its marginal means and standard deviations, and the correlations among all pairs of components. See, for instance, Johnson (1987).

To illustrate how variate generation is done, suppose that $k = 2$ so that we need bivariate normal random vector (Z_1, Z_2) with correlation $r = r_{12}$. If W_1, W_2 are i.i.d. $\mathrm{N}(0,1)$ random variables, then the following transformation works:

$$\begin{pmatrix} Z_1 \\ Z_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ r & \sqrt{1-r^2} \end{pmatrix} \begin{pmatrix} W_1 \\ W_2 \end{pmatrix}. \tag{6.35}$$

Notice that by definition Z_1 is $\mathrm{N}(0,1)$, while Z_2 is normally distributed because it is a linear combination of normally distributed random variables. Also, $\mathrm{Var}(Z_2) = r^2 + \left(\sqrt{1-r^2}\right)^2 = 1$. Exercise 37 asks you to prove that $\mathrm{Corr}(Z_1, Z_2) = r$.

Equation (6.35) is based on the fact that the conditional distribution of Z_2 given Z_1 is $\mathrm{N}(rZ_1, 1-r^2)$. This insight for $k = 2$ extends to $k > 2$: The conditional distribution of Z_h given $Z_1, Z_2, \ldots, Z_{h-1}$ is normal, for $h = 2, 3, \ldots, k$. The following algorithm is equivalent to generating Z_1, then generating Z_2 given Z_1, then Z_3 given Z_1, Z_2, etc. (Johnson, 1987):

```
1. Let C be the Cholesky decomposition of the correlation matrix R so that
   $\mathbf{C}^\top \mathbf{C} = \mathbf{R}$.
2. Let $\mathbf{W}^\top = (W_1, W_2, \ldots, W_k)$ where the $W_i$ are i.i.d. $\mathrm{N}(0,1)$.
3. Set $\mathbf{Z} = (Z_1, Z_2, \ldots, Z_k)^\top = \mathbf{C}^\top \mathbf{W}$.
```

Remark 6.2. Although not needed to apply the NORTA method, another useful fact is that if $Z \sim \mathrm{N}(0,1)$, then $\mu + \sigma Z \sim \mathrm{N}(\mu, \sigma^2)$. This provides a method for generating multivariate normal random vectors with any means, variances, and correlations by first generating $Z_1, Z_2, \ldots, Z_k$ with the desired correlations, then applying the transformation to achieve the required means and variances.

Solving the Correlation-Matching Problem

The correlation-matching problem is simplified by two insights, one obvious, the other not:

1. For a NORTA vector, the correlation ρ_{ij} is a function of only the base correlation r_{ij}. This is true because

$$\rho_{ij} = \mathrm{Corr}(X_i, X_j) = \mathrm{Corr}\left(F_i^{-1}\left[\Phi(Z_i)\right], F_j^{-1}\left[\Phi(Z_j)\right]\right)$$

   which is only a function of Z_i and Z_j. Thus, the correlation-matching problem decomposes into $k(k-1)/2$ individual matching problems.
2. Under very mild conditions, $\mathrm{Corr}\left(F_i^{-1}\left[\Phi(Z_i)\right], F_j^{-1}\left[\Phi(Z_j)\right]\right)$ is a continuous, nondecreasing function of r_{ij} (Biller & Nelson, 2003). Thus, a straightforward search can be used to find r_{ij}.

To implement the search we need to be able to calculate $\mathrm{Corr}\left(F_i^{-1}\left[\Phi(Z_i)\right], F_j^{-1}\left[\Phi(Z_j)\right]\right)$ for any given r_{ij}. There are essentially two choices: numerical methods or simulation. We present a simulation approach here because of its simplicity; for numerical methods see Cario and Nelson (1998) and Biller and Nelson (2003).

The simulation method is straightforward: Given a pair of marginals F_i and F_j, a correlation ρ_{ij} to match, and a feasible value of r_{ij} to check, generate m independent replications of $X_i = F_i^{-1}[\Phi(Z_i)]$ and $X_j = F_j^{-1}[\Phi(Z_j)]$, estimate their correlation $\widehat{\rho}$, and adjust r_{ij} until $\widehat{\rho} \approx \rho_{ij}$. In the algorithm below we implement a bisection search, and we call a function `rho` to compute the correlation estimate $\widehat{\rho}$. Notice that the number of replications m and error tolerance $\varepsilon > 0$ must be specified; we comment on these choices later.

```
Empirical correlation matching

1. generate $\mathbf{W}_1 = (W_{11}, W_{12}, \ldots, W_{1m})$ and $\mathbf{W}_2 = (W_{21}, W_{22}, \ldots, W_{2m})$, all
   i.i.d. $\mathrm{N}(0,1)$
2. if $\rho < 0$ then set $\ell = -1$ and $u = 0$
   Else set $\ell = 0$ and $u = 1$
3. set $r = \rho_{ij}$
4. set $\widehat{\rho} = \texttt{rho}(r, \mathbf{W}_1, \mathbf{W}_2)$
5. while $|\widehat{\rho} - \rho_{ij}| > \varepsilon$ do

   a. if $\widehat{\rho} > \rho_{ij}$ then set $u = r$
      Else set $\ell = r$
   b. set $r = (\ell + u)/2$
   c. set $\widehat{\rho} = \texttt{rho}(r, \mathbf{W}_1, \mathbf{W}_2)$

6. loop
7. return $r_{ij} = r$
```

The function below computes the correlation estimate given a value of r:

```
$\texttt{rho}(r, \mathbf{W}_1, \mathbf{W}_2)$

1. set $Z_{1j} = W_{1j}, j = 1, 2, \ldots, m$
2. set $Z_{2j} = rZ_{1j} + \sqrt{1 - r^2}\, W_{2j}, j = 1, 2, \ldots, m$
3. set $X_{ij} = F_i^{-1}[\Phi(Z_{ij})], i = 1, 2;\ j = 1, 2, \ldots, m$
4. return
   $$\texttt{rho} = \frac{\sum_{h=1}^{m}(X_{1h} - \bar{X}_1)(X_{2h} - \bar{X}_2)}{\sqrt{[\sum_{h=1}^{m}(X_{1h} - \bar{X}_1)^2]\,[\sum_{h=1}^{m}(X_{2h} - \bar{X}_2)^2]}}$$
```

Clearly, the choice of sample size m and error tolerance ε matter. We recommend m no less than 10,000 and ε no greater than $0.05|\rho_{ij}|$, with $0.01|\rho_{ij}|$ preferred. Chen (2001) describes how a sequence of experiments with increasing m and decreasing ε can be used to provide controlled error. This is essential because as k gets larger, a poorly estimated $\mathbf{R}$ may not be positive definite, a condition required for it to be a legitimate correlation matrix.

Also notice that only one set of $\mathbf{W}_1, \mathbf{W}_2$ needs to be generated.

Evaluation of the CDF and Inverse CDF

The NORTA method as described here requires three things:

1. The capability to evaluate the standard normal cdf, $\Phi(\cdot)$.
2. The capability to evaluate the inverse cdfs of the target marginal distributions, $F_1^{-1}(\cdot), F_2^{-1}(\cdot), \ldots, F_k^{-1}(\cdot)$.
3. The capability to generate i.i.d. standard normal random variates W_{ij}.

Many numerical libraries contain efficient methods to evaluate $\Phi(\cdot)$, and computer code can be found in textbooks such as Bratley et al. (1987). The NORTA method is very sensitive to having an accurate evaluation of the normal cdf, and implementations that do not handle the tails in a sensible way can cause the empirical correlation matching to fail to converge.

As discussed in Sect. 6.4, even when $F^{-1}(\cdot)$ cannot be expressed in closed form, it is still possible to numerically invert the cdf. Numerical methods to evaluate the inverse cdf of many common distributions can also be found in numerical libraries as well as Bratley et al. (1987).

Exercises

1. Suppose that $Z_1, Z_2, \ldots$ are i.i.d. positive random variables, and $X = \prod_{i=1}^n Z_i$. Let $W = \ln(X) = \sum_{i=1}^n \ln(Z_i)$, and suppose that $\mathrm{E}(\ln(Z_1)^2) < \infty$. Use the central limit theorem and the continuous mapping theorem to show that X is asymptotically lognormal.
2. Show that the hazard function for the Weibull distribution is increasing for $\alpha > 1$, decreasing for $\alpha < 1$, and constant for $\alpha = 1$.
3. Derive the relationship between the mean and variance of Y and Y' when $Y = a + (b-a)Y'$.
4. Show that X and X' in (6.9) have the same skewness and kurtosis.
5. What problem could occur if instead of solving the least-squares formulation in (6.10), we instead tried to solve
$$\widehat{\theta}_U = \mathrm{argmin}_\theta \sum_{i=1}^m w(i) \left(F(X_{(i)}; \theta) - \frac{i}{m} \right)^2 ?$$
Hint: Suppose F is a distribution with unbounded support, such as the exponential distribution.
6. Suppose we want to parameterize a triangular distribution by specifying the minimum, maximum, and mean value. Derive the most likely value b given this information.
7. Prove that the ecdf is asymptotically consistent $\widehat{F}(x) \xrightarrow{a.s.} F(x)$.

8. Suppose that $\widehat{X} \sim \widehat{F}$, the ecdf of $X_1, X_2, \ldots X_m$. Show that
$$\mathrm{Var}\left(\widehat{X} \middle| X_1, \ldots, X_m\right) = \frac{1}{m}\sum_{i=1}^{m}(X_i - \bar{X})^2.$$
9. For the linearly interpolated ecdf, show that $E(\tilde{F}(x)) \neq F_X(x)$ in general (a single example will suffice).
10. Suppose that $\tilde{X} \sim \tilde{F}$, the linearly interpolated ecdf. Derive a general expression for $\mathrm{E}(\tilde{X}|X_1, \ldots, X_m)$ to show that it is not equal to $\bar{X}$, except as $m \to \infty$.
11. Suggest ways to model the arrival of fans to a basketball game, or more generally arrivals to an event with a fixed starting time and capacity.
12. In this chapter we focused on nonparametric estimators of $\lambda(t)$ or $\Lambda(t)$. However, there is a substantial literature on parametric models, such as
$$\lambda(t) = \exp\left\{\sum_{i=0}^{p}\beta_i t^i + \eta \sin(\omega t + \phi)\right\}$$
(Lee et al., 1991), where p, β_i, η, ω, and ϕ are parameters to be estimated from arrival data. This model allows cyclic behavior via $\eta \sin(\omega t + \phi)$ and a long-term trend through $\sum_{i=0}^{p}\beta_i t^i$. What is the advantage of using this form, rather than the apparently simpler $\lambda(t) = \sum_{i=0}^{p}\beta_i t^i + \eta \sin(\omega t + \phi)$?
13. Prove that $\bar{\Lambda}(t) \xrightarrow{a.s.} \Lambda(t)$ and that $\mathrm{E}\left(\bar{\Lambda}(t)\right) = \Lambda(t)$ for any fixed $t \in [0, T)$. *Hint*: For any fixed t, $N_1(t), N_2(t), \ldots, N_k(t)$ are i.i.d. random variables.
14. For the arrival-rate function $\widehat{\lambda}(t)$ obtained by counting arrivals in fixed intervals of size δ, derive the corresponding integrated rate function.
15. Data on the arrival of faxes, by hour, for 1 month (31 days) can be found on the book website in `FaxCounts.xls`. Use these data to estimate arrival rates for a piecewise-constant, nonstationary arrival process.
16. For the call center in Exercise 16 of Chap. 4, the arrival rate is not actually a steady 60/h, it varies throughout the day. Data on call counts, by hour, for 1 month (31 days) can be found on the book website in `CallCounts.xls`. Use these data to estimate arrival rates for a piecewise-constant, nonstationary Poisson arrival process. Implement this arrival process in your call center simulations. Does this affect your recommendation?
17. Consider the call arrival data from the previous exercise. Let $N(t)$ represent the cumulative number of arrivals by time t. If the process is nonstationary Poisson, then $\mathrm{Var}(N(t))/\mathrm{E}(N(t)) = 1$ for all t, or stated differently $\mathrm{Var}(N(t)) = \beta\mathrm{E}(N(t))$ with $\beta = 1$. Since you have arrival count data, you can estimate $\mathrm{Var}(N(t))$ and $\mathrm{E}(N(t))$ at $t = 1, 2, \ldots, 8$ h. Use these data to fit the regression model $\mathrm{Var}(N(t_i)) = \beta\mathrm{E}(N(t_i))$ and see if the estimated value of β supports the choice of a nonstationary Poisson arrival process. *Hints*: This is regression through the origin. Also, remember that $N(t_i)$ represents the *total number of arrivals by time* t_i.
18. Each summer a car dealer runs a 40-h sale. Customer arrival times (in hours) from two of these sales can be found on the book website in

`ArrivalTimes.xls`. Use these data to form a linearly interpolated integrated rate function $\widehat{\Lambda}(t)$, and then generate customer arrival times for 1 day from a nonstationary Poisson arrival process using $\widehat{\Lambda}(t)$.

19. For the Weibull distribution with cdf (6.4), derive the inverse cdf in a form useful for random-variate generation.
20. Derive the cdf of the triangular distribution (6.15), and then derive the inverse cdf in a form useful for random-variate generation.
21. Show that the inverse cdf for the linearly interpolated cdf is

$$\tilde{X} = X_{(i)} + (m-1)(X_{(i+1)} - X_{(i)}) \left(U - \frac{i-1}{m-1} \right),$$

where $i = \lceil (m-1)U \rceil$.

22. A simple model of a doctor's office is a single server queue. Suppose that patient arrivals to the office are schedule at 15-min intervals, but patients deviate from their scheduled time by an amount that can be modeled as normally distributed with mean -5 min and standard deviation 1 min. How can you generate arrivals to such a queue?
23. We claim the following algorithm implements the inverse cdf method to generate Poisson random variates with mean λ; that is $\Pr\{X = x\} = \mathrm{e}^{-\lambda}\lambda^x/x!$ for $x = 0, 1, 2, \ldots$.

```
a. Set $c = d = \mathrm{e}^{-\lambda}$ and $X = 0$
b. Generate $U \sim U(0,1)$
c. Until $U \le c$ do
     i. $X = X + 1$
    ii. $d = d\lambda/x$
   iii. $c = c + d$
   Loop
d. Return $X$
```

First prove that this algorithm does as claimed. Then derive the expected number of times that the loop is executed for each random variate generated. Notice that it is an increasing function of λ, so the algorithm becomes less and less efficient as λ increases. *Hint*: $\Pr\{X = x\} = \lambda \Pr\{X = x-1\}/x$ for $x \ge 1$.

24. How well does the method of Eq. (6.29) approximate a normally distributed random variate?
25. Prove that $\lambda \int_0^t (1 - G(s))\,\mathrm{d}s = G(t)$ if $G(t) = 1 - \mathrm{e}^{-\lambda t}$. Then show that $G_e \neq G$ in general.
26. Show that if we apply the inversion method to $\bar{\Lambda}(t)$ that (a) only the observed arrival times $T_{(i)}, i = 0, 1, \ldots, C$ can be generated, and (b) that we are likely to generate multiple arrivals with the same arrival time.
27. A disadvantage of using $\widehat{\Lambda}(t)$ is the need to store all of the arrival times $T_{(i)}, i = 1, 2, \ldots, C$. Suppose instead that we only stored $\bar{\Lambda}(\delta), \bar{\Lambda}(2\delta), \ldots, \bar{\Lambda}(m\delta)$, where

$\delta = T/m$. Propose an interpolated estimator of $\Lambda(t)$ based on these values, and derive a variate-generation algorithm for it.

28. For the inversion algorithm applied to $\widehat{\Lambda}(t)$, show that $\tilde{S}_n = C/k$ maps into an arrival at time T.
29. Suppose that arrivals occur according to a NSPP process with integrable rate function $Z\lambda(t)$, conditional on a positive random variable Z with $\mathrm{E}(Z) = 1$. In other words, to simulate this process we first generate a value of Z, and then generate arrivals from a NSPP with rate $Z\lambda(t)$ using inversion or thinning. Derive $E(N(t))$ and $\mathrm{Var}(N(t))/E(N(t))$ for this process, and compare it to results for inverting a renewal process. This arrival process is called a *doubly stochastic Poisson process*, and it is popular in modeling call center arrivals.
30. Let X be a discrete-valued random variable that takes values $x_0 < x_1 < \cdots < x_n$ with probabilities $p_0, p_1, \ldots, p_n$, respectively. Derive, and prove the correctness of a rejection algorithm that first generates an index $I \in \{0, 1, \ldots, n\}$ using the discrete uniform distribution. What is the expected number of trials for your algorithm to generate one X? *Hint*: The following simple algorithm works but is very inefficient: (i) Generate $W \sim U(0,1)$ and set $I = \lceil (n+1)W \rceil - 1$. (ii) Generate $U \sim U(0,1)$. (iii) If $U \le p_I$ then return $X = x_I$; otherwise go to Step (iii). First prove that this algorithm works, and that it has expected number of trials $n+1$. Then derive a better algorithm by applying the concept of a majorizing function as used in the continuous case.
31. Apply your answer to Exercise 30 to produce a rejection algorithm for the binomial distribution on n trials with probability of success $0 < p < 1$. That is,

$$p_X(i) = \Pr\{X = i\} = \binom{n}{i} p^i (1-p)^{n-i},\ i = 0, 1, 2, \ldots, n.$$

Hint: The mode of the binomial distribution is either $i^\star = \lfloor (n+1)p \rfloor$ or $i^\star = \lfloor (n+1)p \rfloor - 1$.

32. In the rejection method, why is it necessary to resample both U and Y on each trial? Is the following algorithm sufficient?

    (a) Generate $U \sim U(0,1)$
    (b) Generate $V \sim g$
    (c) If $U \le f_X(V)/m(V)$, then return $X = V$; else go to Step (b).

33. Consider generating a NSPP with rate $\lambda(t)$ using the following algorithm (with $S_0 = 0$):

$$S_{n+1} = S_n - \ln(1 - U_{n+1})/\lambda(S_n),$$

where $U_1, U_2, \ldots$ are i.i.d. $U(0,1)$. That is, we generate an exponential interarrival time using the arrival rate in force at the most recent arrival time. Prove that this method does not guarantee an arrival process with rate $\lambda(t)$. *Hint*: Consider an arrival rate $\lambda(t)$ that is 0 for some interval of time.

34. The inversion method for generating a nonstationary arrival process requires an equilibrium base process with rate 1 and variance σ_A^2. For $\sigma_A^2 > 1$, Ger-

hardt and Nelson (2009) suggest a balanced hyperexponential distribution; this means that $\tilde{A}$ is exponentially distributed with rate λ_1 with probability p, and exponentially distributed with rate λ_2 with probability $1-p$. "Balance" means that $p/\lambda_1 = (1-p)/\lambda_2$. Thus, there are only two free parameters, p and λ_1. Show that we achieve the desired arrival rate and variance if

$$p = \frac{1}{2}\left(1 + \sqrt{\frac{\sigma_A^2 - 1}{\sigma_A^2 + 1}}\right) \qquad \lambda_1 = 2p.$$

35. The inversion method for generating a nonstationary arrival process requires an equilibrium base process with rate 1 and variance σ_A^2. When $\sigma_A^2 < 1$, Gerhardt and Nelson (2009) suggest a mixture of Erlangs of common order. First find an integer k such that $1/k \le \sigma_A^2 < 1/(k-1)$. Then with probability p, $\tilde{A}$ is Erlang distributed with $k-1$ phases and mean $(k-1)/\lambda$, and with probability $1-p$ it is Erlang distributed with k phases and mean k/λ. Thus, there are two free parameters, p and λ. Show that we achieve the desired arrival rate and variance if

$$\begin{aligned} p &= \frac{1}{1+\sigma_A^2}\left(k\sigma_A^2 - \sqrt{k(1+\sigma_A^2) - k^2\sigma_A^2}\right) \\ \lambda &= k - p. \end{aligned}$$

36. Suppose Y is a continuous valued random variable with strictly increasing cdf F. Prove that $U = F(Y)$ has the $U(0,1)$ distribution. This is known as the *probability–integral transformation*.
37. Show that the transformation (6.35) implies that $\mathrm{Corr}(Z_1, Z_2) = r$. *Hint*: Since we have already shown that $\mathrm{Var}(Z_1) = \mathrm{Var}(Z_2) = 1$, then $\mathrm{Corr}(Z_1, Z_2) = \mathrm{Cov}(Z_1, Z_1)$.
38. For a random variable X that has density function

$$f(x) = \begin{cases} 0, & x < -1 \\ \dfrac{x+1}{2}, & -1 \le x \le 1 \\ 0, & x > 1 \end{cases}$$

provide two variate-generation algorithms, one using the inverse cdf and the other using rejection.
39. As part of the assembly process for laptop computers, batches of computers are loaded into a rack and software is copied to their hard drives. The time to load computers into the rack depends on how many computers are in the batch, but is not perfectly predictable. For a simulation of laptop assembly, you need an input model that generates a load time given a batch size. Develop an appropriate input modeling and variate-generation algorithm, carefully justifying your approach. You have the following data:

Batch size	Load time (min)
5	9.9
10	15.2
15	20.2
20	24.4
10	14.4
10	14.8
15	20.8
5	10.0
5	9.0
15	19.5
5	9.9
5	9.4
10	15.8
15	20.1
20	25.7
5	10.5
10	15.6
10	15.2
5	10.2
20	24.5

40. Given observations of arrival times on $[0,T]$ the linearly interpolated integrated rate function is

$$\widehat{\Lambda}(t) = \left(\frac{C}{C+1}\right)\left[\frac{i}{k} + \frac{1}{k}\left(\frac{t-T_{(i)}}{T_{(i+1)}-T_{(i)}}\right)\right], \text{ when } T_{(i)} < t \le T_{(i+1)}$$

with $T_{(0)} = 0$, $T_{(C+1)} = T$, and $T_{(i)}$ the sorted arrival times. Suppose that we wanted to generate arrivals with this integrated rate function, but by thinning rather than inversion. Thus, we need an arrival-rate function $\widehat{\lambda}(t)$. First give an expression for the rate function that is implied by $\widehat{\Lambda}(t)$. Then give an expression for the maximum arrival rate $\tilde{\lambda}$ to use for thinning.

41. Implement the random-vector example from Sect. 6.3.2 and compare your results to those in the book which used $m = 10{,}000$ and $\varepsilon = 0.01|\rho_{12}|$. Try fitting various marginal distributions for which you have access to the inverse cdf (Excel, Matlab, R, and other software applications have the inverse cdf of many distributions as built-in functions, along with the cdf of the normal distribution).

42. A popular full-period MCG is $z_i = 16{,}807 z_{i-1} \bmod 2^{31} - 1$. Implement this generator, using double-precision floating point arithmetic to avoid overflow problems.

43. Starting with $z_0 = 1$, generate ten additional starting seeds 100,000 values apart for the generator in Exercise 42 above.

44. Show that a MRG of order K can be represented as

$$\mathbf{z}_i = \mathbf{A}\mathbf{z}_{i-1} \bmod m,$$

where $\mathbf{z}_i = (z_{i-K+1}, z_{i-K}, \ldots, z_i)^\top$ and

$$\mathbf{A} = \begin{pmatrix} 0 & 1 & \cdots & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & \cdots & 1 \\ a_k & a_{k-1} & \cdots & a_1 \end{pmatrix}.$$

Also show that we can skip ahead q values by using the relationship

$$\mathbf{z}_{i+q} = \mathbf{A}^q \mathbf{z}_{i-1} \bmod m = (\mathbf{A}^q \bmod m)\mathbf{z}_{i-1} \bmod m.$$

45. For a MCG show that $z_i = a^i z_0 \bmod m$. *Hint*: If b_1, b_2, m are integers, and $r_i = b_i \bmod m$, then $(r_1 r_2 \bmod m) = (b_1 b_2 \bmod m)$.
46. For the following combined linear congruential generator of the L'Ecuyer type:

$$Z_i = (Z_{i,1} - Z_{i,2}) \bmod 15$$

$$U_i = \begin{cases} \dfrac{Z_i}{16}, & Z_i > 0 \\ \dfrac{15}{16}, & Z_i = 0, \end{cases}$$

where the two generators are

$$\begin{aligned} Z_{i,1} &= (11 Z_{i-1,1}) \bmod 16 \qquad \text{with } Z_{0,1} = 1 \\ Z_{i,2} &= (2 Z_{i-1,2}) \bmod 13 \qquad \text{with } Z_{0,2} = 1 \end{aligned}$$

generate U_1 and U_2 by hand. Next implement the generator.
47. Implement the linearly interpolated quantile method of Sect. 6.2.4 so that it can take as input a data set of X's and a number of quantiles k, and can then generate n values of $\tilde{X}$.
48. Using the algorithm you created for the linearly interpolated quantile method for the previous exercise, examine the impact of the choice of k by doing the following: Generate 10,000 variates from some distribution, such as the standard normal, to be your real-world data X. Compute the sample mean and variance and plot a histogram of this data. Now for a fixed value of k, generate 10,000 values of $\tilde{X}$. Compute the sample mean and variance and plot a histogram of this data; compare it to the corresponding results for X. Try values of $k = 10, 25, 50, 100$.
49. Generalize the linearly interpolated quantile method so that it can employ any set of quantiles q_i, not just equally spaced quantiles $q_i = (i-1)/(k-1)$. Why might we want more closely spaced quantiles in some portion of the range of X?
50. Distribution fitting software is widely available, including standalone products, products integrated into simulation software, and packages or libraries for R and

Python. Find such software. What distributions will it fit? What fitting methods are used? What diagnostic tools are provided?

51. Select a distribution from which you can generate random variates, such as a Weibull distribution, and generate a sample of 10,000 values. Using the distribution fitting software you found for the previous exercise, apply it to the first $10, 30, 50, 100, 500, 1000$ and then all 10,000 observations. Does the software always recommend the correct distribution family? If you force it to fit the correct distribution family, how close are the parameters to the ones you set?

Chapter 7
Simulation Output

This chapter describes simulation output analysis, specifically estimating means, probabilities, and quantiles of simulation output, along with measures of error on these estimates. The distinction between risk and error is emphasized, along with the impact of uncertainty about the input processes that drive the simulation.

7.1 Performance Measures, Risk, and Error

Figure 7.1 displays a histogram and empirical cdf (ecdf) of 1000 replications of the time to complete the project planning stochastic activity network (SAN) described in Sect. 3.4 and simulated in Sect. 4.4. Let the random variable Y denote the time to complete the project with cdf F_Y; $Y_1, Y_2, \ldots, Y_{1000}$ are the results from 1000 i.i.d. replications; $Y_{(1)} \le Y_{(2)} \le \cdots \le Y_{(1000)}$ are the sorted values (called *order statistics*); and $\widehat{F}$ is the ecdf. From these figures we can visualize estimators of several system performance measures:

- The expected value of the time to complete the project (the mean), $\mu = \mathrm{E}(Y)$, is estimated by the sample mean $\bar{Y} = \sum_{i=1}^{n} Y_i/n$.
- The probability that the project completes in no more than 5 days, $\theta = F_Y(5)$, is estimated by $\widehat{F}(5) = \#\{Y_i \le 5\}/1000$.
- The 0.95 quantile of the time to complete the project, $\vartheta = F_Y^{-1}(0.95)$, is estimated by $\widehat{F}^{-1}(0.95) = Y_{(950)}$. The 0.95 quantile is a time ϑ such that the probability of completing the project by or before that time is 0.95.

This section covers estimating and interpreting these performance measures based on simulation output.

For the SAN, the probability θ and the quantile ϑ are more relevant performance measures than the mean μ. For whoever requested the project, θ is the probability

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_7

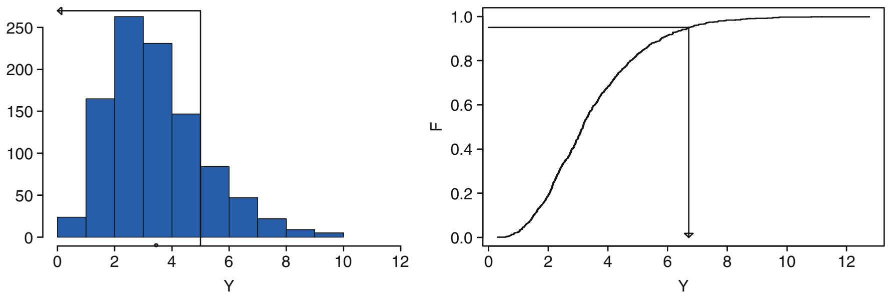

Fig. 7.1 Histogram (*left*) and ecdf (*right*) of 1000 replications of the time to complete the stochastic activity network. Histogram: Project completion times ≤ 5 are indicated by the *arrow*; the sample mean by the *circle*. Ecdf: $Y(950)$ is indicated by the *arrow*

that it completes by a desired date. For whoever is executing the project, ϑ is a completion date that they can promise with a given probability of being able to meet it. The mean time to complete the project is less informative: Assuming that the project will only be undertaken once, the histogram indicates that a completion time near the mean is possible but not at all certain.

By way of contrast, for the hospital reception described in Sect. 3.2 and simulated in Sect. 4.3, the focus was on μ, the long-run average (mean) patient or visitor waiting time. The mean is clearly relevant here as it summarizes performance averaged over the very large number of patients and visitors who will arrive over time. Less obvious is what measures are most relevant for the Asian option described in Sect. 3.5 and simulated in Sect. 4.5. While it may seem similar to the SAN example, since we will likely only buy or sell this option once, financial theory shows that under certain assumptions about the market, the price of an option should be its expected payoff under a particular probability model; see, for instance, Glasserman (2004, Sect. 1.2). Therefore, valuing the option by its expected value ν makes sense.

A point of emphasis in this chapter is that some performance measures, such as the mean μ, are most useful in characterizing a system over the long run, where "long run" implies a large number of occurrences. Other measures, such as a probability θ or quantile ϑ can be used as measures of *risk*, helping to quantify what is likely to happen "the next time" rather than "over the long run." Of course, μ, θ, and ϑ all have to be estimated, and estimates are subject to sampling error. Measures of sampling error are another theme of this chapter; they indicate whether we have done enough simulation (e.g., obtained enough replications) to trust the performance estimates we have. This chapter focuses on measures of error for estimators that are functions of i.i.d. data from across replications; results for dependent data from a single replication of a steady-state simulation are addressed in Chap. 8.

7.1.1 Point Estimators and Measures of Error

Point estimators that are sample means, or functions of sample means, were analyzed in Sect. 5.2. By way of review, and keeping the SAN data in Fig. 7.1 in mind, if we want to estimate $\mu = \mathrm{E}(Y)$ from i.i.d. observations with finite variance σ^2, then the natural unbiased estimator is the sample mean $\bar{Y}$ which has standard error $\mathrm{se}(\bar{Y}) = \sigma/\sqrt{n}$. For large n the central limit theorem justifies the $(1-\alpha)100\%$ confidence interval (CI) for μ

$$\bar{Y} \pm z_{1-\alpha/2}\frac{S}{\sqrt{n}}, \tag{7.1}$$

where S is the sample standard deviation, computed as the square root of the sample variance

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \bar{Y})^2 \tag{7.2}$$

$$= \frac{1}{n-1}\left[\sum_{i=1}^{n} Y_i^2 - \frac{1}{n}\left(\sum_{j=1}^{n} Y_j\right)^2\right]. \tag{7.3}$$

The expression in Eq. (7.3) is the one that is most often implemented because it allows S^2 can be computed in one pass through the data, while (7.2) requires two passes.

For the data in Fig. 7.1 a 95% CI for μ is 3.46 ± 0.11. The ± 0.11 is a measure of error for our point estimate of the mean time to complete the project, 3.46 days. If we increased the number of replications we would expect the width of the CI to decrease as $1/\sqrt{n}$.

Estimates of probabilities such as $\theta = \Pr\{Y \le y\}$ can also be viewed as sample means, since the ecdf evaluated at any fixed value of y is the average of indicator functions:

$$\widehat{F}(y) = \frac{1}{n}\sum_{i=1}^{n} I(Y_i \le y). \tag{7.4}$$

Thus, for large n the CI (7.1) still applies and careful algebra shows that

$$S^2 = \left(\frac{n}{n-1}\right)\widehat{F}(y)\left(1 - \widehat{F}(y)\right). \tag{7.5}$$

When n is large the ratio $n/(n-1)$ is often treated as 1.

For the data in Fig. 7.1 a 95% CI for θ is 0.17 ± 0.02. Again, the ± 0.02 is a measure of error for our point estimate of the probability that the project completes in 5 days or less, 0.17. In this case we actually know that $\theta = 0.165329707$, well within the measure of error.

The standard error of a probability estimator $\widehat{\theta}$ is

$$\mathrm{se}(\widehat{\theta}) = \sqrt{\frac{\theta(1-\theta)}{n}}$$

which is maximized at $\theta = 1/2$. However, for probabilities relative error is more revealing:

$$\frac{\mathrm{se}(\widehat{\theta})}{\theta} = \sqrt{\frac{1-\theta}{n\theta}}.$$

As $\theta \to 0$ the size of the estimation error relative to the probability that we are estimating increases dramatically; in other words, the more rare the event is, the more difficult it is to estimate its probability precisely.

Remark 7.1. In the discussion above we assumed that $\theta \neq 0$. See Louis (1981) for the case when the event of interest is never observed to occur on any observation of the simulation experiment.

Suppose that the cdf of Y, F_Y, is strictly increasing and has a density f_Y. The q-quantile is the value ϑ such that $F_Y(\vartheta) = q$, or equivalently $\vartheta = F_Y^{-1}(q)$. Clearly the natural estimator $\widehat{\vartheta} = \widehat{F}^{-1}(q) = Y_{(\lceil nq \rceil)}$ is not a sample average, so its distributional properties are more complicated. Nevertheless it can be shown that $\widehat{\vartheta}$ does satisfy a central limit theorem:

$$\sqrt{n}\left(\widehat{\vartheta} - \vartheta\right) \xrightarrow{D} \mathrm{N}\left(0, \frac{q(1-q)}{[f_Y(\vartheta)]^2}\right) \tag{7.6}$$

as $n \to \infty$ provided $f_Y(\vartheta) > 0$.[1] Thus, a large n approximation of the standard error is

$$\mathrm{se}\left(\widehat{\vartheta}\right) \approx \sqrt{\frac{q(1-q)}{n[f_Y(\vartheta)]^2}}.$$

Unfortunately, even though the standard error of $\widehat{\vartheta}$ decreases as $1/\sqrt{n}$, like a sample average, this result is not easily applied since it involves the unknown density function f_Y evaluated at the unknown quantile ϑ. However, this result is helpful for seeing the impact of q on the standard error, as shown in the following example.

Suppose $f_Y(y) = \mathrm{e}^{-y}, y \geq 0$, the exponential distribution with mean 1. Then $\vartheta = -\ln(1-q)$, so that

$$\mathrm{se}\left(\widehat{\vartheta}\right) \approx \sqrt{\frac{q(1-q)}{n\left[\exp\left(\ln(1-q)\right)\right]^2}} = \sqrt{\frac{q}{n(1-q)}}.$$

Thus, the standard error of $\widehat{\vartheta} = Y_{(\lceil nq \rceil)}$ as an estimator of the q quantile increases dramatically as q approaches 1, meaning a quantile farther out in the right tail. For instance, when Y is exponentially distributed the standard error of $\widehat{\vartheta}$ for estimating the 0.99 quantile is roughly ten times larger than the standard error for estimating the

[1] This central limit theorem is obtained by noting that $\Pr\{\sqrt{n}(\widehat{\vartheta} - \vartheta) \leq y\} = \Pr\{\widehat{\vartheta} \leq \vartheta + y/\sqrt{n}\} = \Pr\{\widehat{F}(\vartheta + y/\sqrt{n}) > q\}$ and then using the fact that the central limit theorem for averages applies to $\widehat{F}$.

0.5 quantile (the median), for the same n. Therefore, the more extreme the quantile is the more difficult it is to estimate.

Fortunately, there is a CI for quantiles that is not based on the central limit theorem. Notice that since $\Pr\{Y_i \leq \vartheta\} = q$, and the observations Y_i are i.i.d., then

$$\#\{Y_i \leq \vartheta\} \sim \text{Binomial}(n,q).$$

This implies that if we find integers $0 \leq \ell < u \leq n$ such that

$$\Pr\{Y_{(\ell)} \leq \vartheta < Y_{(u)}\} = \sum_{i=\ell}^{u-1} \binom{n}{i} q^i (1-q)^{n-i} \approx 1-\alpha,$$

then $[Y_{(\ell)}, Y_{(u)}]$ forms an approximate $(1-\alpha)100\%$ CI for ϑ. In words, we find (ℓ, u) such that the probability that at least $n-u+1$ of the largest Y_i's are greater than ϑ, and at least ℓ of the smallest Y_i's are less than or equal to ϑ, is approximately $1-\alpha$.

When n is large and neither nq nor $n(1-q)$ is small, the normal approximation to the binomial distribution gives easy approximations for ℓ and u:

$$\begin{aligned} \widehat{\ell} &= \left\lfloor nq - z_{1-\alpha/2}\sqrt{nq(1-q)} \right\rfloor \\ \widehat{u} &= \left\lceil nq + z_{1-\alpha/2}\sqrt{nq(1-q)} \right\rceil . \end{aligned} \tag{7.7}$$

For instance, with $q = 0.95$, $n = 1000$ and $z_{0.975} = 1.96$, an approximate 95% confidence interval for the 0.95 quantile is given by $[Y_{(936)}, Y_{(964)}]$. Notice that the values of $\ell = 936$ and $u = 964$ are independent of the actual outputs. For the data in Fig. 7.1, the point estimate is 6.71 days with 95% CI $[6.43, 7.05]$.

7.1.2 Measures of Risk and Error

The distinction between risk and error in simulation output analysis is important, and often misunderstood.

- *Measures of risk directly support decision-making.* If $F_Y(5)$, which is the probability of completing the SAN project in 5 days or less, is too small, then we might decide not to undertake the project, or to commit additional resources to insure its completion. Similarly, if the 0.95 quantile of completion time, $F_Y^{-1}(0.95)$, is too many days then we might not be willing to bid on the project. The standard deviation of project completion time, σ, is also a measure of risk since it quantifies the average deviation of the actual project completion time from its mean.
- *Measures of error directly support experiment design.* They tell us if we have expended enough simulation effort (e.g., replications) to be confident in our estimates of system performance.

Figure 7.2 displays a MORE (measure of risk and error) plot of the first 100 replications of the SAN output from Fig. 7.1. The MORE plot adds to the histogram

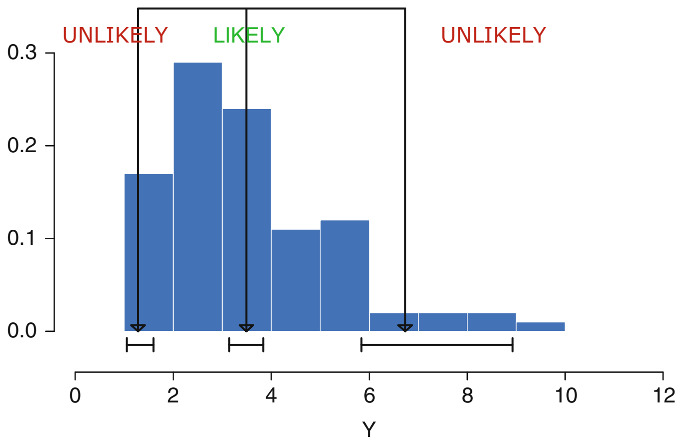

Fig. 7.2 MORE plot of first 100 replications of time to complete the SAN project

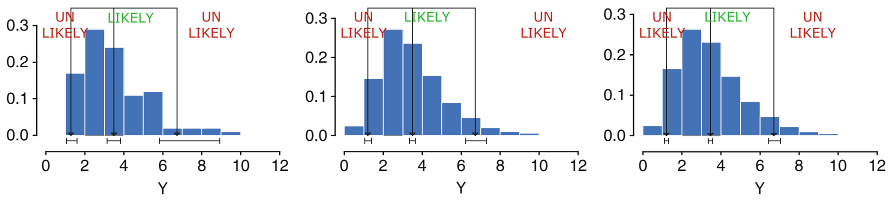

Fig. 7.3 MORE plots for the SAN simulation with (from *left* to *right*) $n = 100, 500, 1000$ replications

a central arrow indicating the sample average, and two other arrows indicating the 0.05 and 0.95 quantile estimates (other quantiles could be used). Thus, the box over the histogram contains 90% of the simulated outcomes that are most central, and so it is labeled as "Likely." The areas beyond the box are labeled "Unlikely" indicating that they are possible, but more extreme.

Of course, the sample mean and sample quantiles are only estimates of the true mean and quantiles, so the MORE plot also includes confidence intervals for each of these displayed as intervals just below the arrowheads. For instance, Fig. 7.2 shows that the true 0.95 quantile could have substantial error and we should be cautious about making decisions based on it.

Figure 7.3 shows a sequence of MORE plots of the SAN data moving from $n = 100$ to $n = 500$ to $n = 1000$ replications. Notice that the error in the estimates of the mean and the quantiles decreases, but risk remains, which illustrates that we can simulate away error, but not risk. This plot is useful for displaying both risk and error in an intuitive way. See Nelson (2008) for additional details on the MORE plot.

7.2 Input Uncertainty and Output Analysis

The output analysis methods described in the previous section implicitly assumed that the performance parameters of the simulation model (e.g., the 0.95 quantile of the simulated time to complete the project) are also the performance parameters of the real-world system in which we are interested (e.g., the real-world project); therefore they only measure estimation error.

Section 5.1 described other sources of error that arise in simulation modeling and analysis, including *input uncertainty*, which refers to the unknown difference between the input models used in the simulation and those that best characterize the real-world system. We focus on the case when the input models have been derived from real-world data, as described in Sects. 6.1–6.3. We first illustrate that input uncertainty can be significant, and then provide one way to capture it.

7.2.1 Input Uncertainty: What Is It?

Consider an $M/M/\infty$ queue, which is a special case of the parking lot problem described in Sect. 3.1 when the arrival rate does not change over time, but instead is a constant λ customers per time. We again denote the mean service time by τ. Much is known about this queue, in particular if Y denotes the steady-state number of customers in the system, then Y has a Poisson distribution with mean $\lambda\tau$.

Suppose λ and τ are not known, so we observe m i.i.d. interarrival times, $A_1, A_2, \ldots, A_m$, and m i.i.d. service times $X_1, X_2, \ldots, X_m$ from the "real world" and use them to fit input models. Specifically, we estimate λ by

$$\widehat{\lambda} = \left(\frac{1}{m}\sum_{i=1}^{m} A_i\right)^{-1}$$

and τ by

$$\widehat{\tau} = \frac{1}{m}\sum_{i=1}^{m} X_i.$$

That is, we estimate the arrival rate as the inverse of the sample mean time between arrivals, and the mean service time by the sample mean service time. We assume that we are confident that the distributions of A and X are exponential, but are uncertain about the values of the parameters.

To allow for an analysis, suppose that when we simulate this queue we record a single observation of Y (the number in the queue) in steady state on each of n replications, say $Y_1, Y_2, \ldots, Y_n$ (typically we would observe $Y(t)$ for some period of time and take a time average). We then estimate the steady-state mean number in queue by the sample mean

$$\bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i.$$

Clearly $\mathrm{E}\left(\bar{Y}|\widehat{\lambda},\widehat{\tau}\right) = \widehat{\lambda}\widehat{\tau}$ and $\mathrm{Var}(\bar{Y}|\widehat{\lambda},\widehat{\tau}) = \widehat{\lambda}\widehat{\tau}/n$. Why? Because conditional on $\widehat{\lambda}$ and $\widehat{\tau}$ each observation Y is Poisson with mean $\widehat{\lambda}\widehat{\tau}$, and the mean equals the variance for a Poisson random variable. However, since the input parameters are estimates, we can be almost sure that $\widehat{\lambda} \neq \lambda$ and $\widehat{\tau} \neq \tau$; that is, our parameter estimates are not the true real-world values. What is the impact of this input uncertainty?

In the appendix to this chapter we show that

$$\mathrm{E}(\bar{Y}) = \frac{m}{m-1}\lambda\tau \tag{7.8}$$

$$\begin{aligned}\mathrm{Var}(\bar{Y}) &= \frac{m}{n(m-1)}\lambda\tau + \frac{m(2m-1)(\lambda\tau)^2}{(m-1)^2(m-2)} \\ &\approx \frac{\lambda\tau}{n} + \frac{2(\lambda\tau)^2}{m},\end{aligned} \tag{7.9}$$

where the mean and variance are with respect to both the distributions of $\widehat{\lambda}$ and $\widehat{\tau}$ (the input uncertainty) and the Poisson distribution of the simulation output (the estimation error). From Eqs. (7.8)–(7.9) we see that $\bar{Y}$ is biased. Also, its variance has two components, a familiar one that depends on the number of simulation replications n, and an unfamiliar one that depends on the amount of real-world data m used to "fit" the input processes. In this example the bias is slight even with a modest sample of input data (about 1% if we have $m = 100$ real-world observations). However, the variance due to input uncertainty (the second term in (7.9)) could easily dominate the simulation variance; worse, this source of variance is not mathematically derivable for realistic problems. In the next section we provide a simple method to obtain a rough estimate of the variance due to input uncertainty.

7.2.2 *Input Uncertainty: What to Do*

Estimation error is something we control through simulation effort (e.g., the number of replications, n). Input uncertainty, on the other hand, depends on the quantity of real-world data we have and it may be impossible or impractical to obtain more. So our goal will be to estimate how large input uncertainty is relative to estimation error; the larger it is the more care we should take in using the simulation results. In terms of the example of the previous section, and in particular the variance decomposition in Eq. (7.9), we would like to know if the term $2(\lambda\tau)^2/m$, which corresponds to input uncertainty, is large relative to $\lambda\tau/n$, which represents simulation error.

To simplify the explanation, suppose for the moment that our simulation model has a single input distribution, F_X, from which we have an i.i.d. sample of real-world data $X_1, X_2, \ldots, X_m$. The true input distribution F_X will be estimated by $\widehat{F}$, a function of this data, which might be the empirical distribution or a parametric distribution with unknown parameters as in the $M/M/\infty$ example of the previous section.

We now represent the output of the simulation on replication j, implemented using estimated input distribution $\widehat{F}$, as

$$Y_j = \mu(\widehat{F}) + \varepsilon_j, \tag{7.10}$$

where $\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n$ are i.i.d. with mean 0 and variance σ_S^2 representing the natural variability from replication to replication in the simulation. The mean term, $\mu(\widehat{F})$, depends on what input model we actually used in the simulation.

For instance, in the $M/M/\infty$ example we simulated the service times as exponential with mean $\widehat{\tau}$ when what we really wanted was exponential with mean τ; and the arrival process as Poisson with rate $\widehat{\lambda}$ when it should be λ. So for the example $\mu(\widehat{F}) = \widehat{\lambda}\widehat{\tau}$, which depends on the random sample of real-world data.

Suppose that instead of one sample of real-world data, we had b independent samples, $X_{i1}, X_{i2}, \ldots, X_{im}, i = 1, 2, \ldots, b$, and from the ith sample we estimated an input model $\widehat{F}_i$. We could then imagine simulating n replications using *each* input model, leading to the output

$$Y_{ij} = \mu(\widehat{F}_i) + \varepsilon_{ij} \tag{7.11}$$

for $i = 1, 2, \ldots, b$ and $j = 1, 2, \ldots, n$.

Model (7.11) is known as a *random-effects model* (see, e.g., Montgomery, 2009), because the mean term $\mu(\widehat{F}_i)$ is random, in this case depending on the particular sample $X_{i1}, X_{i2}, \ldots, X_{im}$ that we happened to use to estimate F_X. We can characterize input uncertainty by the value of $\sigma_I^2 = \mathrm{Var}[\mu(\widehat{F}_i)]$, which is analogous to the second term in (7.9). A big simplifying approximation we make is that σ_S^2, the variance of the simulation output, does not depend on which $\widehat{F}_i$ we have; this will rarely be precisely true, but will be acceptable to get an approximation of the effect of input uncertainty.

For Model (7.11) an estimator of σ_I^2 is

$$\widehat{\sigma}_I^2 = \frac{\widehat{\sigma}_T^2 - \widehat{\sigma}_S^2}{n}, \tag{7.12}$$

where

$$\widehat{\sigma}_T^2 = \frac{n}{b-1} \sum_{i=1}^{b} (\bar{Y}_{i\cdot} - \bar{Y}_{\cdot\cdot})^2$$

and

$$\widehat{\sigma}_S^2 = \frac{1}{b(n-1)} \sum_{i=1}^{b} \sum_{j=1}^{n} (Y_{ij} - \bar{Y}_{i\cdot})^2$$

(Montgomery, 2009). A "$\cdot$" subscript indicates averaging over that index. Intuitively, $\widehat{\sigma}_T^2$ measures both input and simulation variability, so by subtracting out simulation variability we are left with variance due to input uncertainty. In Exercise 17 you are asked to show that $\mathrm{E}\left(\widehat{\sigma}_I^2\right) = \sigma_I^2$ under Model (7.11). The ratio $\widehat{\sigma}_I^2/(\widehat{\sigma}_S^2/n)$ is a measure of how large input uncertainty is relative to estimation error.

The practical problem with this proposal is the need for multiple real-world samples. And if we really had this extra data we would almost certainly use all mb observations together to better estimate F_X. However, a statistical technique known as *bootstrapping* allows us to imitate the effect of having multiple real-world samples, even though we only have one (see Efron and Tibshirani (1993) for a general reference on bootstrapping).

Again let $\{X_1, X_2, \ldots, X_m\}$ be the one real-world sample of data. To implement the procedure above, we generate b bootstrap samples, each of size m, by sampling m times *with replacement* from $\{X_1, X_2, \ldots, X_m\}$. As is typical in the bootstrapping literature, we denote the ith bootstrap sample as $X_{i1}^\star, X_{i2}^\star, \ldots, X_{im}^\star$. We then fit an input distribution $\widehat{F}_i^\star$ to this bootstrap sample, using the same method we used with the real-world sample, and implement the procedure above by letting $\widehat{F}_i^\star$ stand in for $\widehat{F}_i$. In a problem with more than one input model, such as the $M/M/\infty$ which has two, we do bootstrapping for each distribution separately. Notice that there is no requirement that we have equal numbers of real-world observations for each input model.

Here is the procedure in algorithm form:

```
1. Given real-world data $\{X_1, X_2, \ldots, X_m\}$, do the following:
2. For $i$ from 1 to $b$
    a. Generate the bootstrap sample $X_{i1}^\star, X_{i2}^\star, \ldots, X_{im}^\star$ by sampling $m$ times
       with replacement from $\{X_1, X_2, \ldots, X_m\}$.
    b. Fit $\widehat{F}_i^\star$ to $X_{i1}^\star, X_{i2}^\star, \ldots, X_{im}^\star$. (If there is more than one input model, do
       Steps 2a and 2b for each one.)
    c. Simulate $n$ replications $Y_{ij}, j = 1, 2, \ldots, n$ using input model(s) $\widehat{F}_i^\star$.
3. Estimate $\sigma_I^2$ using (7.12).
```

The value of b should be at least 10. The number of replications n should be reasonable for the simulation problem if we were not evaluating input uncertainty.

As an example, consider the $M/M/\infty$ queue from the previous section. If $\lambda = 5$ customers/min, $\tau = 1$ min, we observe $m = 100$ real-world interarrival times and $m = 100$ real-world service times, and we make $n = 10$ replications of the simulation, then Eq. (7.9) gives

$$\mathrm{Var}(\bar{Y}) \approx \frac{\lambda\tau}{n} + \frac{2(\lambda\tau)^2}{m} = \frac{5}{10} + \frac{50}{100} \approx \frac{\sigma_S^2}{n} + \sigma_I^2,$$

where the last $\approx$ holds if our random-effects model is approximately correct. Running the procedure with $b = 100$ bootstrap samples we obtained $\widehat{\sigma}_S^2 = 5.321$ and $\widehat{\sigma}_I^2 = 0.546$, both very close to the correct values of 5 and 0.5, respectively. Since $\widehat{\sigma}_S^2/10 = 0.5321$, we see that input uncertainty is approximately as large as estimation error. Whether or not this is a concern depends on whether or not a standard error of $\sqrt{0.5321} = 0.729$ is considered large or small for this problem, since the overall error is approximately double this.

Remark 7.2. Because we estimate σ_I^2 by subtracting in Eq. (7.12) it is possible that the estimate will be negative even though variances cannot be negative. A negative $\widehat{\sigma}_I^2$ should be interpreted as implying that the variability due to input uncertainty is small relative to simulation error. See Ankenman and Nelson (2012) for additional discussion and also a method for identifying which input models account for the most input uncertainty.

The method for assessing input uncertainty variance presented here is the simplest we know that illustrates the key ideas. There are other methods that do not require additional bootstrap experiments, as well as methods that obtain a more precise estimate of σ_I^2. For surveys see Lam (2016) and Song and Nelson (2017).

Like the $M/M/\infty$ queue, the typical simulation has multiple, maybe many, input models. Therefore, natural questions to ask are "Which of the input models are the largest contributors to σ_I^2?" and "For which inputs would obtaining more real-world data reduce σ_I^2 the most?" These are not simple questions because the contributions of the various input models are not, in general, additive as illustrated by the second term in Eq. (7.9) for the $M/M/\infty$ queue. Nevertheless, most approaches to assessing the individual contributions treat them as approximately additive.

Again let $\widehat{F}$ represent the estimated input models. The standard variance decomposition gives

$$\mathrm{Var}(\bar{Y}(n)) = \mathrm{E}\left[\mathrm{Var}(\bar{Y}(n) \mid \widehat{F})\right] + \mathrm{Var}\left[\mathrm{E}(\bar{Y}(n) \mid \widehat{F})\right].$$

In our notation the "input uncertainty" is $\sigma_I^2 = \mathrm{Var}[\mathrm{E}(\bar{Y}(n)|\widehat{F})]$. Suppose $\widehat{F}$ consists of p input models $\widehat{F} = \{\widehat{F}_1, \widehat{F}_2, \ldots, \widehat{F}_p\}$; $p = 2$ for the $M/M/\infty$ example. Let m_j be the number of real-world observations used to estimate the jth input model. The essential idea is to further expand the approximation as

$$\mathrm{Var}(\bar{Y}) \approx \frac{\sigma_S^2}{n} + \sigma_I^2 \approx \frac{\sigma_S^2}{n} + \frac{\sigma_{I1}^2}{m_1} + \frac{\sigma_{I2}^2}{m_2} + \cdots + \frac{\sigma_{Ip}^2}{m_p},$$

where σ_{Ij}^2/m_j is the (approximate) contribution of the jth input model, the so-called *first-order effect* $\mathrm{Var}[\mathrm{E}(\bar{Y}(n)|\widehat{F}_j)]$. Under some conditions this approximation can be justified. Different methods take different approaches to estimate the terms on the right-hand side. Two approaches that build on bootstrap experiments as described in this section are Lam and Qian (2018) and Song and Nelson (2015).

7.2.3 Input Uncertainty vs. Sensitivity Analysis

Properties of a simulation output Y, such as its mean, probabilities and quantiles, depend on many aspects of the simulation model, including the chosen input distributions, the values of the distributions' parameters (for parametric distributions), and the values of controllable structural variables such as the number of servers at each station in a queueing network. Some of these quantities are uncertain—e.g., the input-model parameters when they were estimated from data—while others are not—e.g., the number of servers assigned to each station in the queueing network. In either case one might be interested in how properties of Y are influenced by changes in these quantities.

Input uncertainty, local sensitivity analysis and global sensitivity analysis all address some aspect of this "influence" although the boundaries between them are a bit fuzzy. The purpose of this section is to explain informally what each tries to accomplish, deferring to Chap. 9 a mathematical treatment of local sensitivity analysis because of its connection to simulation optimization.

Very loosely,

$$\text{Input Uncertainty} = \text{System Sensitivity} \times \text{Input Model Uncertainty}.$$

When input models are estimated there is uncertainty about the correct distribution. However, whether or not this uncertainty propagates to the outputs depends on how sensitive Y is to this input, with the sensitivity either magnifying or dampening the influence. For instance, due to having only a small sample of real-world data we may have a great deal of uncertainty about the distribution of an input X, but if Y is not a function of X—that is, it is insensitive to X—then the input-distribution uncertainty does not transfer to the output. Some approaches for estimating input uncertainty variance are based on combining an estimate of sensitivity with an estimate of input-parameter variance.

To contrast local and global sensitivity analysis we focus on a particular property of the output Y, its mean $\mathrm{E}(Y)$. Local sensitivity is typically a partial derivative of $\mathrm{E}(Y)$ with respect to some quantity in the simulation at a nominal value. A common case is $\partial \mathrm{E}(Y)/\partial\theta$ at $\theta = \theta_0$, where θ is an input model parameter. For instance, in the $M/M/\infty$ illustration, Y is the steady-state number of customers in the system, and the input parameters are the arrival rate λ and mean service time τ. For this simple model we know that $\mathrm{E}(Y) = \lambda\tau$, and thus $\partial \mathrm{E}(Y)/\partial\lambda = \tau$. Therefore, if we run a simulation with nominal settings $\lambda_0 = 5$ and $\tau_0 = 1$, the local sensitivity is 1, which is interpreted as the increase in the mean number in the system for each unit increase in the arrival rate when the service rate is $\tau_0 = 1$. Notice that local sensitivity does not depend upon how certain we are that, say, $\lambda = 5$ and $\tau = 1$; it simply measures how large a change in $\mathrm{E}(Y)$ can be expected with a small change in λ around these values.

Global sensitivity analysis is widely applied to computer experiments in which there is no inherent randomness, but there are parameters that are not known with certainty. Computer experiments are often deterministic simulations such as numerically integrating a system of differential equations through time when some of the coefficients of the differential equations are uncertain. To assess which parameters have the most influence on the output or response, probability distributions are *imposed* on the parameters; this induces variance in the output of the deterministic simulation (a different set of parameters leads to a different deterministic output). Global sensitivity analysis tries to partition this variance among the parameters as a measure of their influence; first-order effects are a standard approach. Clearly there is a relationship to input uncertainty, but the methods are different because there is no inherent system randomness, and the imposed distributions are usually intended to represent lack of knowledge rather than sampling variability due to estimation from data. A good general reference is Saltelli et al. (2008).

Appendix: Input Uncertainty for the $M/M/\infty$ Queue

Here we derive Eqs. (7.8) and (7.9). We will use the following results repeatedly: Suppose $E_1, E_2, \ldots, E_m$ are i.i.d exponential each with rate ν (mean $1/\nu$), and let $G = \sum_{i=1}^m E_i$. Then G has an Erlang distribution (which is a special case of a gamma distribution), and $G^{-1} = 1/G$ has an inverse gamma distribution. Thus, $m\widehat{\tau}$ has an Erlang distribution, while $\widehat{\lambda}/m$ has an inverse gamma distribution. For these distributions it is known that

$$\begin{aligned}
\mathrm{E}(G) &= \frac{m}{\nu} \\
\mathrm{Var}(G) &= \frac{m}{\nu^2} \\
\mathrm{E}(G^{-1}) &= \frac{\nu}{m-1} \text{ for } m > 1 \\
\mathrm{Var}(G^{-1}) &= \frac{\nu^2}{(m-1)^2(m-2)} \text{ for } m > 2.
\end{aligned}$$

These results imply that $\mathrm{E}(G^2) = \mathrm{Var}(G) + \mathrm{E}^2(G) = m(m+1)/\nu^2$ and $\mathrm{E}(G^{-2}) = \mathrm{Var}(G^{-1}) + \mathrm{E}^2(G^{-1}) = \nu^2/((m-1)(m-2))$. Therefore, the expected value of the point estimator is

$$\begin{aligned}
\mathrm{E}(\bar{Y}) &= \mathrm{E}\left[\mathrm{E}\left(\bar{Y}|\widehat{\lambda}, \widehat{\tau}\right)\right] \qquad (7.13) \\
&= \mathrm{E}\left[\widehat{\lambda}\widehat{\tau}\right] \\
&= \mathrm{E}\left[\widehat{\lambda}\right]\mathrm{E}\left[\widehat{\tau}\right] \\
&= \mathrm{E}\left[\frac{m}{\sum_{i=1}^m A_i}\right] \times \tau \\
&= \frac{m}{m-1}\lambda\tau.
\end{aligned}$$

Notice that in Eq. (7.13) the inner expectation averages over the simulation output distribution, while the outer expectation averages over the possible real-world samples. This result shows that the point estimator is, in general, biased when we must estimate the input models.

To derive the variance, we use the variance decomposition

$$\operatorname{Var}(\bar{Y}) = \operatorname{E}\left[\operatorname{Var}(\bar{Y}|\widehat{\lambda},\widehat{\tau})\right] + \operatorname{Var}\left[\operatorname{E}(\bar{Y}|\widehat{\lambda},\widehat{\tau})\right].$$

Recall that $\operatorname{E}\left(\bar{Y}|\widehat{\lambda},\widehat{\tau}\right) = \widehat{\lambda}\widehat{\tau}$ and $\operatorname{Var}\left(\bar{Y}|\widehat{\lambda},\widehat{\tau}\right) = \widehat{\lambda}\widehat{\tau}/n$. Thus, using our previous result for the mean we get

$$\operatorname{E}\left[\operatorname{Var}(\bar{Y}|\widehat{\lambda},\widehat{\tau})\right] = \operatorname{E}\left[\frac{\widehat{\lambda}\widehat{\tau}}{n}\right] = \frac{m}{n(m-1)}\lambda\tau.$$

The second term is

$$\begin{aligned}
\operatorname{Var}\left[\operatorname{E}(\bar{Y}|\widehat{\lambda},\widehat{\tau})\right] &= \operatorname{Var}\left[\widehat{\lambda}\widehat{\tau}\right] \\
&= \operatorname{E}\left[(\widehat{\lambda}\widehat{\tau})^2\right] - \operatorname{E}\left[\widehat{\lambda}\widehat{\tau}\right]^2 \\
&= \operatorname{E}\left[\widehat{\lambda}^2\right]\operatorname{E}\left[\widehat{\tau}^2\right] - \left(\frac{m}{m-1}\right)^2(\lambda\tau)^2 \\
&= \frac{m^2\lambda^2}{(m-1)(m-2)} \times \frac{(m+1)\tau^2}{m} - \left(\frac{m}{m-1}\right)^2(\lambda\tau)^2.
\end{aligned}$$

The result then follows by collecting terms.

Exercises

1. Show that Eq. (7.5), which gives the sample variance of a probability estimator, is correct.
2. Suppose we observe i.i.d. simulation outputs $Y_1, Y_2, \ldots, Y_n$, each with mean $\mu < \infty$ and variance $\sigma^2 < \infty$, and our goal is to estimate σ^2 using the usual sample variance S^2 given in (7.2). Assuming also that $\operatorname{E}(Y^k) < \infty, k = 3,4$, show that S^2 satisfies a central limit theorem. *Hint*: This is not an easy problem, so start with the easier problem of proving that

$$\tilde{S}^2 = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \mu)^2$$

   satisfies a central limit theorem, then express S^2 as a function of $\tilde{S}^2$. You will need the tools from Sect. 5.2.4.
3. For the ecdf $\widehat{F}$ based on n observations $Y_1, Y_2, \ldots, Y_n$, show that $\widehat{F}^{-1}(q) = Y_{(\lceil nq \rceil)}$ for $0 < q \le 1$.
4. Derive (and perhaps refine) the normal approximation (7.7) used to form a confidence interval for a quantile.

5. Recall the stochastic activity network simulation of Sect. 4.4. Estimate the 0.95 quantile of the time to complete the project and put a 90% confidence interval on it. Use at least 1000 replications.
6. Recall the Asian option simulation of Sect. 4.5. Estimate the 0.95 quantile of its value and a 90% confidence interval on it. Use at least 10,000 replications.
7. Recall the Asian option simulation of Sect. 4.5 where the volatility was $\sigma^2 = (0.3)^2$. With all other parameters unchanged, examine the impact of volatility on the value of the option for $\sigma = 0.1, 0.2, 0.3, 0.4, 0.5$. Estimate the value to within 1% relative error.
8. There are many types of options in addition to the Asian option described in Sect. 3.5 and simulated in Sect. 4.5. A *lookback call option* has the payoff

$$X(T) - \min_{0 \le t \le T} X(t)$$

which can be thought of as the profit at time T from simultaneously buying the asset at its lowest price during the period 0 to T and selling it for the final price at time T. Thus, its value is

$$\mathrm{E}\left[e^{-rT}\left(X(T) - \min_{0 \le t \le T} X(t)\right)\right].$$

Using the same asset model as the Asian option with $T = 1$, $X(0) = \$50$, $r = 0.05$ and $\sigma^2 = (0.3)^2$, estimate the value of this option to within 1% relative error. Notice that you will have to use a discrete approximation for $\min_{0 \le t \le T} X(t)$ taking steps of size $\Delta t = T/m$. Experiment with $m = 8, 16, 32, 64, 128$ steps to see how it effects the estimated value, and provide intuition for the effect you see.
9. There are many types of options in addition to the Asian option described in Sect. 3.5 and simulated in Sect. 4.5. A *down-and-out call option* with barrier B and strike price K has the payoff

$$I\left\{\min_{0 \le t \le T} X(t) > B\right\}(X(T) - K)^+$$

which means that if the value of the asset falls below \$$B$ before the option matures then the option is worthless. Thus, its value is

$$\mathrm{E}\left[e^{-rT} I\left\{\min_{0 \le t \le T} X(t) > B\right\}(X(T) - K)^+\right].$$

Using the same asset model as the Asian option with $T = 1$, $X(0) = \$50$, $K = \$55$, $r = 0.05$ and $\sigma^2 = (0.3)^2$, estimate the value of this option to within 1% relative error. Notice that you will have to use a discrete approximation for $\min_{0 \le t \le T} X(t)$ taking steps of size $\Delta t = T/m$. Experiment with $m = 8, 16, 32, 64, 128$ steps and barrier $B = 30, 35, 40, 45$ to see how they effect the estimated value, and provide intuition for the effect you see.

10. A small warehouse supplies paper products. Trucks pull up to the loading bays, the driver places an order and then waits for the order to be gathered by a warehouse worker with a forklift truck. There are currently four loading bays, and when they are full, trucks wait in the parking lot next to the warehouse.

    The warehouse is open from 7 a.m. to 4 p.m. Trucks arrive throughout the day. The warehouse has two forklift drivers who pick orders one-at-a-time, on a first-come-first-served basis. The truck drivers load their own orders after the entire order is picked (for safety reasons), and then leave the loading dock. As soon as the forklift driver has picked the order for a truck they are free to start on another truck if there is one waiting in a loading bay (i.e., they do not have to wait for the driver to load the order).

    The warehouse company is interested in how much improvement in efficiency they can obtain from three changes: (a) adding another forklift and driver; (b) adding another loading bay; or (c) doing both (a) and (b). The least expensive option is (a), but they might be willing to do (b) or (c) if there is enough value.

    "Efficiency" will be measured in terms of the daily mean number of trucks waiting in the parking lot, mean time a truck spends waiting in the parking lot, mean time a truck waits for a forklift after entering a loading bay, utilization of the bays and the forklifts, and the mean number of trucks still in the parking lot at 4 p.m. (trucks that arrive before 4 p.m. will be served, so having a large number of trucks waiting at 4 p.m. implies that the company will have to pay a great deal of overtime).

    Trucks arrive according to a Poisson arrival process with mean time between arrivals of 18 min. The time to pick orders by the forklift drivers is modeled as Erlang with mean 40 min and four phases; the time for a driver to load the order is modeled as Erlang with mean 12 min and three phases.

    Simulate the system to evaluate the improvement that can be attained from the three options relative to the current set-up (so you also need to simulate the current system). Determine a number of replications so that the relative error on each estimate is no more than 5%. Stop each replication at 4 p.m. and do not worry about the trucks remaining in the system at that time, other than recording the value of the number still in the parking lot.

11. Arrivals to the warehouse in Exercise 10 are not stationary. On the book website in `TruckCounts.xls` you will find hourly arrival counts for 30 days. Use these data to estimate arrival rates for a piecewise-constant, nonstationary Poisson arrival process. Implement this arrival process in your warehouse simulation, rerun each case (current system, one forklift, one loading bay, and both a forklift and loading bay) and report the results.

12. Consider the truck arrival data from the previous problem. Let $N(t)$ represent the cumulative number of arrivals by time t. If the process is nonstationary Poisson, then $\mathrm{Var}(N(t))/\mathrm{E}(N(t)) = 1$ for all t, or stated differently $\mathrm{Var}(N(t)) = \beta\mathrm{E}(N(t))$ with $\beta = 1$. Since you have arrival count data, you can estimate $\mathrm{Var}(N(t))$ and $\mathrm{E}(N(t))$ at $t = 1, 2, \ldots, 9$ h. Use these data to fit the regression model $\mathrm{Var}(N(t_i)) = \beta\mathrm{E}(N(t_i))$ and see if the estimated value of β

supports the choice of a nonstationary Poisson arrival process. *Hints*: This is regression through the origin. Also, remember that $N(t_i)$ represents the *total number of arrivals by time* t_i.

13. FirstTreatment, Inc., plans to open for-profit health care clinics throughout the United States. Your job is to help FirstTreatment estimate the costs to operate such facilities for different patient loads, which will be part of their business plan. The primary continuing cost for a health care clinic is staffing, which will be your focus. There are requirements for providing timely treatment, but having more staff and equipment than needed makes a facility expensive to operate. This exercise is based on an example in Kelton et al. (2011).

FirstTreatment's facility design consists of five stations: Sign-In/Triage; Registration; Examination; Trauma; and Treatment. Each of these are staffed separately.

All patients enter at Sign-in/Triage, where a staff member quickly assesses their condition. Patients with severe injuries or illness are sent to Trauma; once their condition is stabilized they move to Treatment; then they are discharged or transported to a hospital.

Patients with less severe problems proceed from Sign-In/Triage to Registration, and then to Examination for evaluation. After the evaluation, roughly 40% are immediately discharged; the others go to Treatment before they are discharged.

The following distributions have been chosen to represent patient contact time:

- **Sign-in/Triage**: Exponential with mean 3 min
- **Registration**: Lognormal with mean 5, and variance 2 min
- **Examination**: Normal with mean 16 and variance 3 min
- **Trauma**: Exponential with mean 90 min
- **Treatment**:
  - For trauma patients, Lognormal with mean 30 and variance 4 min
  - For non-trauma patients, Lognormal with mean 13.3 and variance 2 min

Patient arrivals will be modeled as a Poisson arrival process, and the movement of patients between stations is so brief that it will be treated as 0.

Staffing levels at each station will clearly depend on patient load, percentage of trauma patients, and the patient service-level requirements. FirstTreatment has asked you to look at a range of options for load and trauma (these are distinct scenarios, not random quantities). The percentage of trauma patients depends very much on where the health care facility is located; it could be as low as 8% or has high as 12% of all patients. For load they are guessing as low as 75 to as high 225 patients per day, on average. FirstTreatment clinics will be open from 6 a.m. to 12 a.m. (18 h); once the doors close at midnight patients currently being treated will be completed, but new arrivals will be directed to a local hospital.

Clearly the most important requirement is that trauma patients be stabilized quickly. FirstTreatment has specified that Sign-in/Triage should be "very fast" and that the wait to see a trauma doctor should average less than 5 min. Average delays of 15–20 min are acceptable for other areas of the clinic.

Simulate the health care clinic and provide FirstTreatment with staffing levels required to cover the range of loads for the clinics they might open. Make enough replications so that you can be confident that the service-level requirements have been met. Provide some insight into whether slightly relaxing the service requirements might reduce staffing cost.

14. Arrivals to the clinic in Exercise 13 are not stationary. On the book website in `ClinicCounts.xls` you will find hourly arrival counts for 30 days for a clinic that expects 225 patients/day on average. Use these data to estimate arrival rates for a piecewise-constant, nonstationary Poisson arrival process. Implement this arrival process in your clinic simulation using the staffing level you found for that load. How well does this staffing level do under the more realistic arrival process?
15. Consider the clinic arrival data from the previous exercise. Let $N(t)$ represent the cumulative number of arrivals by time t. If the process is nonstationary Poisson, then $\mathrm{Var}\left(N(t)\right)/\mathrm{E}\left(N(t)\right) = 1$ for all t, or stated differently $\mathrm{Var}\left(N(t)\right) = \beta\mathrm{E}\left(N(t)\right)$ with $\beta = 1$. Since you have arrival count data, you can estimate $\mathrm{Var}\left(N(t)\right)$ and $\mathrm{E}\left(N(t)\right)$ at $t = 1, 2, \ldots, 18$ h. Use these data to fit the regression model $\mathrm{Var}\left(N(t_i)\right) = \beta\mathrm{E}\left(N(t_i)\right)$ and see if the estimated value of β supports the choice of a nonstationary Poisson arrival process. *Hints*: This is regression through the origin. Also, remember that $N(t_i)$ represents the *total number of arrivals by time t_i*.
16. Consider the $M/M/\infty$ example of Sect. 7.2.1 used to demonstrate the impact of input uncertainty. Suppose that the arrival rate of the Poisson arrival process was estimated in the following alternative way: A time interval $[0, T]$ was observed and λ was estimated by

$$\tilde{\lambda} = \frac{N(T)}{T},$$

where $N(t)$ is the number of arrivals that occurred by time t. This is an unbiased, maximum likelihood estimator of λ if the arrival process is really Poisson. Using this alternative estimator of λ, derive the mean and variance of the simulation estimator described in Sect. 7.2.1.
17. Show that $\mathrm{E}\left(\widehat{\sigma}_I^2\right) = \sigma_I^2$ under Model (7.11).
18. Consider the stochastic activity network where the time to complete the project is

$$Y = \max\{X_1 + X_4, X_1 + X_3 + X_5, X_2 + X_5\}.$$

A sample of data on activity durations for similar activities can be found on the book website in `SANData.xls`. These data are believed to be exponentially distributed but with different means for each activity, means that can be estimated from the data. The goal is to estimate the expected value of the time to complete the project, $\mathrm{E}(Y)$, using 30 replications. Conduct an experiment to also estimate σ_I^2, the additional variance due to input uncertainty.

19. Consider the waiting time of the ith customer in a single-server queue as represented by Lindley's equation

$$Y_i = \max\{0, Y_{i-1} + X_{i-1} - A_i\}$$

with $Y_0 = X_0 = 0$. A sample of data on interarrival times and service times can be found on the book website in `MG1Data.xls`. The interarrival times A are believed to be exponentially distributed and the service times X lognormally distributed; the parameters of these distributions can be estimated from the data. The goal is to estimate the expected value of the time to serve the first 10 customers, $\mathrm{E}\left(\sum_{i=1}^{10} Y_i\right)$, using 30 replications. Conduct an experiment to also estimate σ_I^2, the additional variance due to input uncertainty.

Chapter 8
Experiment Design and Analysis

The purpose of simulation, at least in this book, is to estimate the values of performance measures of a stochastic system by conducting a statistical experiment on a computer model of it. This chapter describes principles and methods for the design and analysis of that experiment.

We have considered several different performance measures, including means, probabilities, and quantiles, sometimes for systems with a natural time horizon, and sometimes for steady-state systems (as time goes to infinity). For this initial overview, let $\theta(\mathbf{x})$ denote a generic performance measure for scenario $\mathbf{x}$. A "scenario" defines a specific instance of a more general system. For example, the scenario variable $\mathbf{x}$ could be a vector of cardinal values (e.g., $\mathbf{x}^\top = (15,9,6,3)$ denotes the number Entry Agents and Specialists assigned before and after noon, respectively, in the fax center simulation of Sect. 4.6) or nominal values (e.g., $\mathbf{x} = 2$ denotes the system design for the hospital reception of Sect. 4.3 that employs an electronic kiosk, while $\mathbf{x} = 1$ denotes the current design with a human receptionist). What we call the scenario variable is sometimes called a decision variable, and having it will be useful when we consider comparing system designs to answer questions such as "What is the probability of a special fax being entered in less than 10 min when we use a staff of $\mathbf{x}^\top = (15,9,6,3)$ agents?"

The reason for conducting a simulation experiment is to produce an estimator of $\theta(\mathbf{x})$, denoted by $\widehat{\theta}(\mathbf{x};T,n,\mathbf{U})$, which is a function of $\mathbf{x}$ and up to three additional quantities:

Stopping time T: The stopping time is the simulation clock time at which a replication ends; it can be a fixed time ("simulate 8 hours, from 8 AM to 4 PM") or a random time ("simulate until the stochastic activity network completes" or "simulate until the system fails"). In some contexts T is called run length. See also the Remark at the end of this section.

Number of replications n: The number of independent and identically distributed replications of the simulation may also be fixed ($n = 30$) or random ("simulate until the width of the confidence interval for $\theta(\mathbf{x})$ is less than 3 minutes").

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_8

(Pseudo)random numbers U: The estimator $\widehat{\theta}(\mathbf{x};T,n,\mathbf{U})$ is a random variable because we treat $\mathbf{U}$ as random numbers. Of course, we will actually use pseudorandom numbers; therefore, we may also think of $\mathbf{U}$ as a fixed set of numbers, say $\mathbf{u}_1$, that can be reused by fixing the starting seed or stream.

Experiment design consists of selecting the scenarios $\mathbf{x}$ *to simulate, controlling the number of replications n to obtain, assigning the pseudorandom numbers* $\mathbf{U}$ *to drive the simulations and, in steady-state simulation, setting the stopping time T.*

Notice that whether or not T is part of the experiment design depends on what we are estimating. For performance measures $\theta(\mathbf{x})$ defined as $T \to \infty$, a choice as to what finite T to use in the simulation is required. Otherwise, T is typically part of the definition of a replication and is not a design choice. Consider the following examples:

Parking lot (*M(t)/M/*∞ queue, Sect. 4.2): Here there was only a single scenario so $\mathbf{x}$ is not needed. One performance measure of interest was

$$\theta = \mathrm{E}\left[\max_{0 \le t \le 24} N(t)\right]$$

the expected value of the maximum number of cars in the lot during a day. Thus, the stopping time $T = 24$ h is part of the definition of the problem, and the primary design decision is choosing the number of replications n.

Hospital reception (*M/G/*1 queue): Here there were two scenarios, the current human receptionist $\mathbf{x} = 1$ and the electronic kiosk $\mathbf{x} = 2$. The performance measure was

$$\theta(\mathbf{x}) = \mathrm{E}[Y(\mathbf{x})],$$

where $Y(\mathbf{x})$ is the steady-state waiting time before service under scenario $\mathbf{x}$. In this case both T and n are design variables, and we may even want a design that specifies $n = 1$ and T very large, such as $T = 10^6$ customers.[1] In addition, if we are interested in $\theta(2) - \theta(1)$, the difference in expected waiting times, then we show in Sect. 9.2 that there are statistical advantages to assigning both simulations the same set of pseudorandom numbers, say $\mathbf{u}_1$. Thus, we could represent the estimator of the difference as

$$\widehat{\theta}(\mathbf{x} = 2; T = 10^6, n = 1, \mathbf{U} = \mathbf{u}_1) - \widehat{\theta}(\mathbf{x} = 1; T = 10^6, n = 1, \mathbf{U} = \mathbf{u}_1).$$

Fax center: Here the scenarios were defined by the number of regular and special agents before and after noon, so $\mathbf{x}^\top = (x_1, x_2, x_3, x_4)$ and there are many possible

[1] Here we have chosen to measure "time" by the number of customers served, rather than via the simulation clock time. Equivalently, we could let T be the simulation time required to serve 10^6 customers.

combinations. If $\mathbf{c}^\top = (c_1, c_2, c_3, c_4)$ is a vector of costs/agent in each of these categories, then the goal of the simulation is

$$\begin{aligned} &\min\ \mathbf{c}^\top \mathbf{x} \\ &\text{subject to:} \\ &\quad \theta_1(\mathbf{x}) \geq 0.96 \\ &\quad \theta_2(\mathbf{x}) \geq 0.80, \end{aligned}$$

where $\theta_1(\mathbf{x})$ and $\theta_2(\mathbf{x})$ are the probabilities that regular and special faxes, respectively, are entered within 10 min of receipt under agent assignment $\mathbf{x}$. The stopping time T is the time when the last fax arriving before 4 p.m. is completed, and is therefore part of the definition of the problem. The number of replications n might be chosen adaptively depending on the variability of the estimators.

Although it is not possible to entirely separate the topics, Sect. 8.1 covers setting the number of replications, n; Sect. 8.2 addresses the choice of stopping time for steady-state simulations, T; and Sect. 8.3 examines a different, but related topic: choice of the estimator, $\widehat{\theta}$. We defer the choice of which scenarios $\mathbf{x}$ to simulate and the assignment of pseudorandom numbers, $\mathbf{U}$ to Chap. 9 on simulation optimization.

Remark 8.1. Although not critical for understanding design and analysis, we use the term *stopping time* in the formal sense of a stopping time of a stochastic process (see, e.g., Karlin and Taylor, 1975). Informally, a stopping time is a point in simulated time, possibly defined by a set of conditions, that can be recognized when it is reached without having to look into the future beyond it. "Simulate for 10 hours of simulation time" defines a stopping time, as does "simulate until 1000 customers have been served" or "simulate until the first fax takes more than 10 minutes to enter." However, "simulate until you reach the maximum number of cars that will be in the garage" is not a stopping time because we have to simulate the entire 24 h to be sure when the maximum occurs.

8.1 Controlling Error via Replications

Classical statistical analysis was developed to infer characteristics of a population (e.g., income of adults living in England) from a small sample of them, and to infer characteristics of a physical entity or process (e.g., yield per acre of corn hybrids) from an expensive or time-consuming scientific experiment. See, for instance, the exceptional history of statistics by Stigler (1986). In both situations data are expensive or time-consuming to obtain, so it is natural to think of the number of observations as small, fixed, and obtained in a one-time study. This leads to an emphasis on tools that extract as much information as possible from little data, and on methods for measuring the remaining estimation error.

Simulation replications, on the other hand, can often be obtained quickly and inexpensively, and since the experiment is carried out on a computer they can also be obtained sequentially rather than in one trial. This can and should lead to designing experiments to *control error* rather than just measure it. We discuss that approach here.

Our goal in this section is to estimate a performance parameter for a single scenario, so we drop the dependence on **x** and denote the parameter by θ. Further, we suppose T is defined by the problem, and since pseudorandom-number assignment is only a design decision when we are comparing scenarios, we also drop the dependence on **U** and T. Therefore, the goal is to estimate $\theta = \mathrm{E}\,[\widehat{\theta}(n)]$. Further, we consider estimators that are averages across n replications, so to emphasize this we replace the notation $\widehat{\theta}(n)$ by $\overline{\theta}(n)$, which is defined to be

$$\overline{\theta}(n) = \frac{1}{n}\sum_{j=1}^{n}\widehat{\theta}_j,$$

where $\widehat{\theta}_j$ is the unbiased estimator of θ from the jth i.i.d. replication. As an illustration, consider the parking lot model of Sect. 4.2 where

$$\theta = \mathrm{E}\left[\max_{0\le t\le 24} N(t)\right].$$

Then $\widehat{\theta}_j$ is the maximum number of cars observed in the parking lot during the jth replication of 24 h.

The normal-theory $(1-\alpha)100\%$ confidence interval (CI) for θ when n is a fixed number of replications is

$$\overline{\theta}(n) \pm t_{1-\alpha/2,n-1}\frac{S(n)}{\sqrt{n}}, \tag{8.1}$$

where

$$S^2(n) = \frac{1}{n-1}\sum_{j=1}^{n}\left(\widehat{\theta}_j - \overline{\theta}(n)\right)^2$$

is the sample variance of the replication estimators. Even when $\widehat{\theta}_1, \widehat{\theta}_2, \ldots, \widehat{\theta}_n$ are not normally distributed, the central limit theorem for averages justifies (8.1) for a large number of replications n because $\sqrt{n}(\overline{\theta}(n) - \theta)$ is asymptotically normal as $n \to \infty$; see Sect. 5.2.2. When we simulated the parking lot for $n = 1000$ replications we got a 95% CI of 237.2 ± 0.7 cars.

One way to think about what a CI does is that it provides an error bound on $|\overline{\theta}(n) - \theta|$ that holds with high confidence; specifically

$$\Pr\left\{|\overline{\theta}(n) - \theta| \le t_{1-\alpha/2,n-1}\frac{S(n)}{\sqrt{n}}\right\} \approx 1-\alpha.$$

But better than just measuring error is controlling it, and specifically reducing it below an acceptable threshold for the decision problem at hand. This is different from increasing the confidence with which the bound holds, which is done through our choice of $1-\alpha$. Instead, we are tightening the bound itself by the choice of n.

The acceptable level of error can be characterized in at least two ways: *Absolute error* means

$$|\overline{\theta}(n) - \theta| \le \varepsilon \tag{8.2}$$

for given ε, while *relative error* means

$$|\overline{\theta}(n) - \theta| \le \kappa|\theta| \tag{8.3}$$

for given $0 < \kappa < 1$. The absolute error ε is in the same units as the performance measure θ (minutes, dollars, cars). For the parking lot simulation we might consider an absolute error of $\varepsilon = 1$ car. On the other hand, κ is a fraction of θ itself. Relative error is often stated as a percentage; for instance, $\kappa = 0.01$ is "1% relative error." For the parking lot simulation it appears that 1% relative error is about 2.4 cars.

An intuitively appealing approach to achieve a given absolute error is to make N replications, where

$$N = \min\left\{n : t_{1-\alpha/2,n-1}S(n)/\sqrt{n} \le \varepsilon\right\}. \tag{8.4}$$

Similarly, to achieve given relative error make N replications where

$$N = \min\left\{n : t_{1-\alpha/2,n-1}S(n)/\sqrt{n} \le \kappa|\overline{\theta}(n)|\right\}. \tag{8.5}$$

Notice that in both cases N is a random variable, so the validity of this approach is not supported by the usual central limit theorem for averages. Instead, theoretical support comes from central limit theorems for $\sqrt{N}\left(\overline{\theta}(N) - \theta\right)$ as $\varepsilon \to 0$ and $\kappa \to 0$, respectively. The classical references are Chow and Robbins (1965) and Nádas (1969). A paper that specifically addresses the simulation setting is Glynn and Whitt (1992); see also Alexopoulos (2006). Intuitively, this works because driving the absolute error ε or the relative error κ to zero drives N to infinity, but the actual proof is more subtle.

However, it is important to understand that, unlike the t-distribution CI (8.1), even when $\widehat{\theta}_1, \widehat{\theta}_2, \ldots$ are normally distributed the bounds (8.2) and (8.3) are not guaranteed to hold with probability $1-\alpha$ when N is chosen in this way. Instead, as we become more demanding (accept smaller and smaller error) these stopping rules are valid.

To implement (8.4) or (8.5) we need to be able to update $\overline{\theta}(n)$ and $S^2(n)$ efficiently as we increase the number of replications. Two relationships help:

$$\overline{\theta}(n) = \overline{\theta}(n-1) + \frac{\widehat{\theta}_n - \overline{\theta}(n-1)}{n} \tag{8.6}$$

$$\sum_{j=1}^{n}\left(\widehat{\theta}_j - \overline{\theta}(n)\right)^2 = \sum_{j=1}^{n-1}\left(\widehat{\theta}_j - \overline{\theta}(n-1)\right)^2 + \left(\widehat{\theta}_n - \overline{\theta}(n)\right)\left(\widehat{\theta}_n - \overline{\theta}(n-1)\right). \tag{8.7}$$

The following algorithm, due to Knuth (1998), implements this recursion.

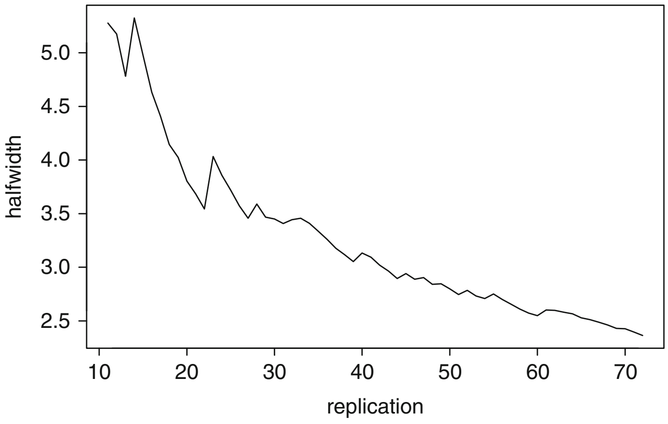

Fig. 8.1 Confidence interval halfwidth as a function of the number of replications for the parking lot simulation with 1% relative error stopping rule

```
1. $n = 0$, mean $= 0$ and sum $= 0$
2. Obtain replication result $\widehat{\theta}_{n+1}$
    a. $n = n + 1$
    b. diff $= \widehat{\theta}_n -$ mean
    c. mean $=$ mean $+$ diff$/n$
    d. sum $=$ sum $+$ diff $* (\widehat{\theta}_n -$ mean$)$
    e. If $n > 1$, then $\overline{\theta}(n) =$ mean and $S^2(n) =$ sum$/(n-1)$
3. Next replication
```

The particular stopping test (8.4) or (8.5) would be inserted in Step 2e, and N is the first n for which it is satisfied. However, there are good reasons not to implement the algorithm exactly that way.

Unfortunately, rules such as (8.4) or (8.5) are subject to stopping too early, before N has a chance to become large. This is because the sample variance $S^2(n)$ can be quite variable for small n, meaning there is a chance that it will be far smaller than the true variance; this leads to premature termination and a CI that is too short. Unless simulation replications are very time-consuming, the following simple guideline will help: Make at least 10 replications before applying the stopping test for the first time, with 60 replications being better. A further advantage of starting at $n = 60$ is that the t quantiles $t_{1-\alpha/2,n-1} \approx z_{1-\alpha/2}$ for $n \geq 60$, so there is no need to obtain the t values.

Figure 8.1 is a plot of the CI halfwidth against the number of replications for the parking lot problem with the 1% relative error stopping rule, computing the confidence interval for the first time at $n = 11$ replications. The simulation stops after $N = 72$ replications with confidence interval 236.306 ± 2.363 cars. Thus, the estimated relative error is $2.363/236.306 \approx 1\%$. Notice that due to sampling variability the halfwidths do not decrease monotonically, and it is this variability that can cause early stopping.

One type of performance measure that does not fit neatly into this framework is a quantile. As discussed in Sect. 7.1.1 the q-quantile of a continuous valued random variable Y is $\vartheta = F_Y^{-1}(q)$, and the standard estimator is $\widehat{\vartheta} = Y_{(\lceil nq \rceil)}$, where $Y_{(1)} \le Y_{(2)} \le \cdots \le Y_{(n)}$ are the sorted values from all n replications. Therefore, we need the results from all n replications to form a single quantile estimator. The same is true of using a sample variance to estimate the $\mathrm{Var}(Y)$.

Because we can often afford to make the number of replications very large, we can adapt the methods in this section to performance measures such as quantiles and variances using an approach called *batching*. The idea is that we redefine the jth output to be a statistic computed from a batch of b individual replications; this statistic can be an order statistic, a sample variance, or any other statistic of interest:

$$\underbrace{Y_1,\ldots,Y_b}_{\widehat{\theta}_1},\underbrace{Y_{b+1},\ldots,Y_{2b}}_{\widehat{\theta}_2},\ldots,\underbrace{Y_{(n-1)b+1},\ldots,Y_{nb}}_{\widehat{\theta}_n}.$$

With this adjustment the methods described in this section can be applied directly to the batch statistics. However, a batch size b needs to be chosen, and in the case of quantile estimation this is particularly critical because quantile estimators are biased, with that bias decreasing in b. Thus, b should not be small.

8.2 Design and Analysis for Steady-State Simulation

In this section we address the challenging problem of steady-state simulation. To do so we will consider the simulation of a single scenario, so we can drop the scenario variable $\mathbf{x}$ and the random-number assignment $\mathbf{U}$ from the notation. Therefore, the goal is to estimate a steady-state parameter θ with an estimator $\widehat{\theta}(T,n)$, since the number of replications n and the run length T are both design choices. We will also introduce an additional design decision that is only relevant for steady-state simulation, an amount of data to delete, d.

For most of the section we will focus on the problem of estimating the steady-state mean μ from a discrete-time output series $Y_1,Y_2,\ldots,Y_m$, so the run length is the number of observations $T = m$ and the estimator will be a sample mean. Therefore, in this section we employ the notation

$$\widehat{\theta}(T,n) = \overline{Y}(n,m,d)$$

as defined below.

We used the $M/G/1$ queue (see Sect. 3.2) as an example of a steady-state simulation, one for which the quantity we want to estimate is defined in the limit as the simulation run length goes to infinity. For the $M/G/1$ queue this suggested that we make a very long simulation run (large number of customers m) and estimate, say, the steady-state mean waiting time μ by the average of the observed waiting times $Y_1, Y_2, \ldots, Y_m$. But we noticed that the waiting times early in the run tend to be smaller than μ because the queue starts empty, which causes $\overline{Y}(m)$ to be biased low. To reduce this effect, we suggested letting the simulation generate waiting times for a while (say d of them) before starting to actually include them in the sample average. That is, we used the truncated sample average

$$\overline{Y}(m,d) = \frac{1}{m-d} \sum_{i=d+1}^{m} Y_i. \tag{8.8}$$

We may also choose to make n replications, yielding n i.i.d. averages $\overline{Y}_1(m,d)$, $\overline{Y}_2(m,d), \ldots, \overline{Y}_n(m,d)$, giving the overall average

$$\overline{Y}(n,m,d) = \frac{1}{n} \sum_{j=1}^{n} \overline{Y}_j(m,d). \tag{8.9}$$

Of course, this includes the special case of $n = 1$ replication. Experiment design and analysis for steady-state simulation focuses on choosing n, m, and d. We now give two precise statements of the "steady-state simulation problem." Neither problem is directly solvable in practice, but knowing which one best represents your objective helps determine the approach you will use.

Fixed-precision problem: Given $\varepsilon > 0$ or $0 < \kappa < 1$, choose n, m and d so that $\sqrt{\mathrm{MSE}\left(\overline{Y}(n,m,d)\right)} \le \varepsilon$ (absolute error), or $\sqrt{\mathrm{MSE}\left(\overline{Y}(n,m,d)\right)}/\mu \le \kappa$ (relative error). That is, choose n, m, and d so that a fixed precision is achieved. This statement of the problem favors large d to effectively eliminate bias, and $n > 1$ replications to make it easier to quantify error via, say, a confidence interval.

Fixed-budget problem: Given a budget of N observations, choose n, m, and d to minimize $\mathrm{MSE}\left(\overline{Y}(n,m,d)\right)$ subject to $nm \le N$ and $d < m$. That is, spend a fixed amount of simulation effort in a way that minimizes MSE. Examination of the asymptotic MSE (8.11) or the AR(1) result (8.12) below makes it immediately obvious that this formulation favors solutions with $n = 1$ replication and $m = N$.

Notice that both formulations focus on $\mathrm{MSE}\left(\overline{Y}(n,m,d)\right)$ as the measure of the quality of the estimator, which makes sense because bias is a feature of steady-state simulation. The next two sections present methods for attacking each of these problems, but to have them make sense we need to understand the properties of the Estimator (8.9).

We argued in Sect. 5.2.3 that when steady-state makes sense then we should expect that the mean squared error of the sample average of one long replication can be approximated by

$$\mathrm{MSE}\left(\overline{Y}(1,m,0)\right) \approx \frac{\beta^2}{m^2} + \frac{\gamma^2}{m}, \tag{8.10}$$

where β is the asymptotic bias and γ^2 is the asymptotic variance. We referred to this quantity as the asymptotic MSE. We can also argue that for large m and d much smaller than m that

$$\mathrm{MSE}\left(\overline{Y}(1,m,d)\right) \approx \frac{\beta(d)^2}{(m-d)^2} + \frac{\gamma^2}{m-d},$$

where $\beta(0) = \beta$. Why should the asymptotic bias be a function of the amount of data deleted, d? Recall that

$$\beta = \sum_{i=1}^{\infty} (\mathrm{E}(Y_i) - \mu)$$

which can be thought of as the area under the bias curve from 0 to ∞, or the total accumulated bias. And since the bias goes to 0 as $1/m$, we approximated the bias of $\overline{Y}(1,m,0)$ as β/m. Similarly, let

$$\beta(d) = \sum_{i=d+1}^{\infty} (\mathrm{E}(Y_i) - \mu)$$

be the total accumulated bias from $d+1$ to ∞, so the bias of $\overline{Y}(1,m,d)$ is approximately $\beta(d)/(m-d)$. Adjusting the asymptotic bias for deletion is important because we expect the bias to be largest early in the replication.

Finally, for general number of replications n,

$$\mathrm{MSE}\left(\overline{Y}(n,m,d)\right) \approx \frac{\beta(d)^2}{(m-d)^2} + \frac{\gamma^2}{n(m-d)}. \tag{8.11}$$

That the variance decreases as $1/n$ is expected. That the bias is unaffected by replications is a consequence of

$$\mathrm{E}\left(\overline{Y}(n,m,d) - \mu\right) = \frac{1}{n}\sum_{j=1}^{n} \mathrm{E}\left(\overline{Y}_j(m,d)\right) - \mu = \mathrm{E}\left(\overline{Y}_1(m,d)\right) - \mu.$$

Replications reduce variance, but not bias, in steady-state simulation.

As a specific example, Exercise 2 asks you to show that the asymptotic MSE of the AR(1) surrogate process is

$$\mathrm{MSE}\left(\overline{Y}(n,m,d)\right) \approx \frac{(y_0-\mu)^2\varphi^{2d+2}}{(m-d)^2(1-\varphi)^2} + \frac{\sigma^2}{n(m-d)(1-\varphi)^2} \tag{8.12}$$

and further to prove that for $\varphi > 0$ the asymptotic squared bias (the first term) is decreasing in d for m large enough. Clearly, the variance $\sigma^2/\left(n(m-d)(1-\varphi)^2\right)$ is increasing in d. Thus, there is a bias-variance trade off in choosing a deletion point d for steady-state simulation.

Expression (8.12) illustrates features of the steady-state simulation problem that affect the MSE: Starting far from steady-state conditions (as represented by $(y_0-\mu)^2$ being large), or having slowly diminishing bias (as represented by $|\varphi|$ close to one) make the bias the dominant feature of MSE, suggesting deletion. A large variance of the outputs (as represented by σ^2 large) makes the variance dominant, suggesting a long-run length m or number of replications n.

8.2.1 Fixed-Precision Problem

The fixed-precision setting makes sense when the simulation budget is not a constraint: typically this means that the simulation model executes fast enough, and that the time until the results are needed is long enough, that we can afford to "simulate until we are done." This is often the case in practice, and it makes no sense to enforce an arbitrary simulation budget when one does not exist.

The approach we take to solve this problem consists of two steps:

1. Determine a run length m and deletion point $\widehat{d}$ such that the bias $\beta(\widehat{d})/(m-\widehat{d})$ is effectively 0. Employ this deletion point to obtain truncated averages that are (effectively) without bias.
2. Measure the remaining statistical error via a confidence interval or standard error, and increase the run length or number of replications until the desired precision has been achieved.

The specific methods applied to achieve Steps 1 and 2 depend on whether we plan to make $n = 1$ or $n > 1$ replications. We focus on the $n > 1$ case; methods described in Sect. 8.2.2 for a fixed budget could be adapted to obtain fixed precision from a single run by letting batches replace replications.

The primary feature that makes steady-state simulation hard is that unlike variance, the bias $\beta(d)/(m-d)$ is not directly estimable. However, a sufficient condition for $\beta(d)/(m-d) \approx 0$ is that $\mathrm{E}(Y_i) - \mu \approx 0$ for all $i > d$, *and this will be true if* $\mathrm{E}(Y_i)$ *is essentially unchanging for* $i > d$. Even though the bias cannot be estimated, the change in $\mathrm{E}(Y_i)$ as a function of i can, particularly if we can afford to make replications. Therefore, we solve Step 1 by estimating the $\mathrm{E}(Y_i)$ vs. i curve and observing where it appears to stop changing.

8.2.1.1 Estimating the Deletion Point

1. Obtain output data $Y_{ij}, i = 1,2,\ldots,m$ and $j = 1,2,\ldots,n$. For instance, in the $M/G/1$ simulation $Y_{3,8}$ is the waiting time of the third customer on the eighth replication. The run length m will be a guess initially, while the number of replications n should be at least 10.
2. Average the ith observation across all n replications:

$$\overline{Y}_i = \frac{1}{n}\sum_{j=1}^{n} Y_{ij}$$

for $i = 1, 2, \ldots, m$. For instance, in the $M/G/1$ simulation $\overline{Y}_3$ would be the average waiting time of the third customer to arrive in each of the n replications.

3. Examine or plot $\overline{Y}_i$ vs. i, or perhaps a smoothed version of it.
   (a) If the data are too variable to detect the trend, then increase the number of replications n.
   (b) If the plot still seems to be trending up or down, then increase the run length m.
   (c) If there is an observation number $\widehat{d}$ such that beyond it the plot seems to be varying consistently around a fixed central value, then choose $\widehat{d}$ as the deletion point.

If done successfully, then we have selected $\widehat{d}$ such that

$$\text{MSE}\left(\overline{Y}_j(m,\widehat{d})\right) \approx 0 + \frac{\gamma^2}{m-\widehat{d}}$$

and therefore

$$\text{MSE}\left(\overline{Y}(n,m,\widehat{d})\right) \approx \text{Var}\left(\overline{Y}(n,m,\widehat{d})\right) \approx \frac{\gamma^2}{n(m-\widehat{d})}.$$

To solve Step 2, we form a confidence interval for μ

$$\overline{Y}(n,m,\widehat{d}) \pm t_{1-\alpha/2,n-1}\frac{S(n,m,\widehat{d})}{\sqrt{n}},$$

where

$$S^2(n,m,\widehat{d}) = \frac{1}{n-1}\sum_{j=1}^{n}\left(\overline{Y}_j(m,\widehat{d}) - \overline{Y}(n,m,\widehat{d})\right)^2$$

is the sample variance of the replication averages. The degrees of freedom analysis presented in the next section suggests that if the confidence interval is not yet short enough to meet the precision requirement, then it makes sense to increase the number of replications if $n < 30$, and otherwise to increase the run length m. However, it is usually more convenient to fix m and just increase the number of replications until the confidence interval is short enough. If this approach is adopted, then make sure that m is substantially larger than d, say $m \approx 10d$.

To achieve fixed absolute error we increase the number of replications n until

$$t_{1-\alpha/2,n-1}S(n,m,\widehat{d})/\sqrt{n} \leq \varepsilon.$$

Similarly, to achieve fixed relative error we increase the number of replications until

$$\frac{t_{1-\alpha/2,n-1}S(n,m,\widehat{d})/\sqrt{n}}{\overline{Y}(n,m,\widehat{d})} \leq \kappa.$$

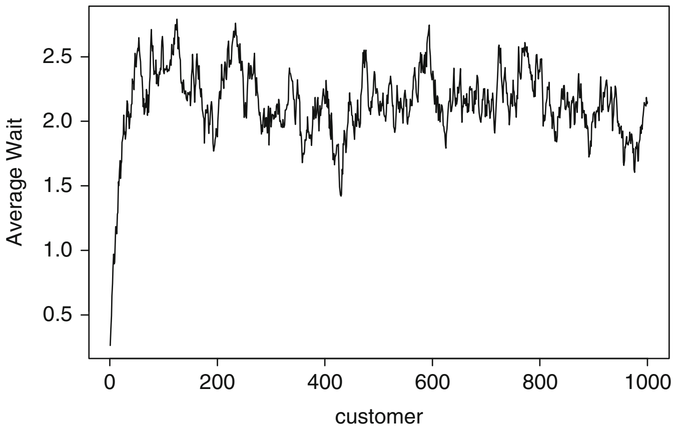

Fig. 8.2 Mean plot, by customer number, averaged across $n = 100$ replications of the $M/G/1$ queue

To illustrate these ideas, recall the $M/G/1$ queueing simulation of Sect. 4.3, where the goal was to estimate the steady-state mean customer waiting time in queue. Suppose we desired this estimate to have no more than 3% relative error.

Figure 8.2 shows the average waiting time, by customer number, for the first $m = 1000$ customers averaged across $n = 100$ replications. The bias due to starting the simulation empty is apparent. At somewhere around 100 customers the average waiting times seem to vary around a central value, so we might take $\widehat{d} = 100$. This is a subjective judgement; additional replications, or smoothing of the plot, would be helpful to confirm this choice, and there is no penalty (other than computation time) for making it larger.

Given $\widehat{d} = 100$, a convenient choice for the run length m is 1100 customers (at least ten times as large). Thus, each replication will generate waiting times for the first $m = 1100$ customers, and report the average of the last $m - \widehat{d} = 1000$ of them. A guess of $n = 500$ replications satisfied the relative error requirement, as it gave

$$\frac{t_{0.975,499}S(500,1100,100)/\sqrt{500}}{\overline{Y}(500,1100,100)} \approx \frac{0.053}{2.154} \approx 0.025 < 0.03.$$

Remark 8.2. This $M/G/1$ queue, which represents hospital reception, is not a very congested system, so the amount of data deletion required is very small. This is fine for a book example, but should not be considered representative of all simulations, or even all queueing simulations. Heavily congested queueing systems may require substantially more data deletion.

8.2.2 Fixed-Budget Problem

When we have a fixed budget of N observations, and no sense whether it is generous or tight, the natural experiment design is to make a single replication ($n = 1, m = N$) to have the best chance of overcoming the bias. We will again have two steps: selection of a deletion point d, and estimation of the remaining error in the point estimator.

8.2.2.1 Deletion with a Fixed Budget

We consider three options for choosing a deletion point d:

No deletion: As shown in Eq. (8.10), the bias component of MSE goes down as $1/m^2$, while variance only diminishes as $1/m$. With no direct information about the bias, deleting no data is a viable option.

Use system knowledge: Although we can treat steady-state simulation as an abstract statistical problem, in applications we are simulating a real or conceptual system that we understand well enough to model it. Insight about how long it might take the real system to "warm up" can be used to set a deletion point. If a factory starts empty, how many hours or days of production would be required to ramp up to normal operations? If all echelons of a supply chain are fully stocked, how many weeks would it take for orders, transportation and inventory levels to start behaving normally? Answers to these questions provide deletion points.

Data-driven deletion point: A number of methods have been proposed for choosing a deletion point based on the output data. These include the following. Note that the first two approaches try to eliminate bias, while the third tries to actually minimize the MSE of the resulting estimator.

- Plot the cumulative sample mean: Although we cannot average across replications to reveal the bias trend, we can plot $\sum_{i=1}^{t} Y_i/t$ vs. t, for $t = 1, 2, \ldots, m$, and look for a point at which it stabilizes around a central value. This is often done because it is easy, but it tends to delete many more observations than necessary since the cumulative average retains all of the most biased observations. In a fixed-budget setting it is not typically a good idea to be overly conservative with respect to bias since it results in a corresponding penalty in terms of estimator variance. Schruben (1982) provides a single-replication plot that is more sensitive to bias, but is less directly useful for determining a deletion point.
- Test for bias: If a deletion point d can be chosen based on knowledge of the system, or even a guess, then statistical tests have been proposed in which the null hypothesis is $H_0 : \mathrm{E}(Y_{d+1}) = \cdots = \mathrm{E}(Y_m)$. See, for instance, Schruben et al. (1983). A description of a number of these tests, and evaluation of their performance, can be found in Cash et al. (1992).

- Estimate the MSE-optimal deletion point: Although the MSE as a function of d cannot be estimated directly, the marginal standard error rule (MSER, White, 1997) minimizes the value of a statistic whose expectation is asymptotically proportional to MSE (Pasupathy & Schmeiser, 2010). Since it attacks the stated problem directly, and is easy to implement, we describe MSER in more detail below.

The $\mathrm{MSER}(d)$ statistic is

$$\mathrm{MSER}(d) = \frac{1}{(m-d)^2} \sum_{i=d+1}^{m} \left(Y_i - \overline{Y}(m,d)\right)^2 \tag{8.13}$$

and the estimated deletion point is chosen to minimize $\mathrm{MSER}(d)$. Specific rules include

$$\widehat{d} = \mathrm{argmin}_{d=0,\ldots,\lfloor m/2 \rfloor} \mathrm{MSER}(d) \tag{8.14}$$

and

$$\tilde{d} = \min\left\{d : \mathrm{MSER}(d) \le \min[\mathrm{MSER}(d-1), \mathrm{MSER}(d+1)]\right\}. \tag{8.15}$$

The choice $\widehat{d}$ minimizes MSER restricted to the first half of the output series, while $\tilde{d}$ yields the first local minimum of MSER; both choices recognize that the MSER statistic is quite variable when d is close to m and try to avoid choosing d too large just because of output variability.

Clearly the bias and variance of the outputs $Y_1, Y_2, \ldots, Y_m$ influence the value of MSER, but it is not immediately obvious how minimizing MSER relates to minimizing MSE. Intuitively, either a strong trend in the mean of the series $Y_1, Y_2, \ldots, Y_m$ (which suggests that the amount of deletion should be large) or substantial marginal variance $\mathrm{Var}(Y_i)$ (which suggests that the amount of deletion should be small) will cause (8.13) to be large, so minimizing MSER balances these contributions. Pasupathy and Schmeiser (2010) showed that, under very general conditions, for every $d = 0, 1, 2, \ldots$

$$\lim_{m\to\infty} \frac{\mathrm{MSE}(\overline{Y}(m,d))}{\mathrm{E}\left(\mathrm{MSER}(d)\right)} = c,$$

where c is a constant that depends on the output process. That is, for large m, MSE and MSER are proportional, in expectation, so that minimizing the expected value of MSER is equivalent to minimizing MSE. Of course, the expected value of MSER is not known, so we minimize the statistic instead.

Exercise 5 asks you to show that

$$\sum_{i=d+1}^{m} \left(Y_i - \overline{Y}(m,d)\right)^2 = \sum_{i=d+1}^{m} Y_i^2 - \frac{1}{m-d}\left(\sum_{i=d+1}^{m} Y_i\right)^2.$$

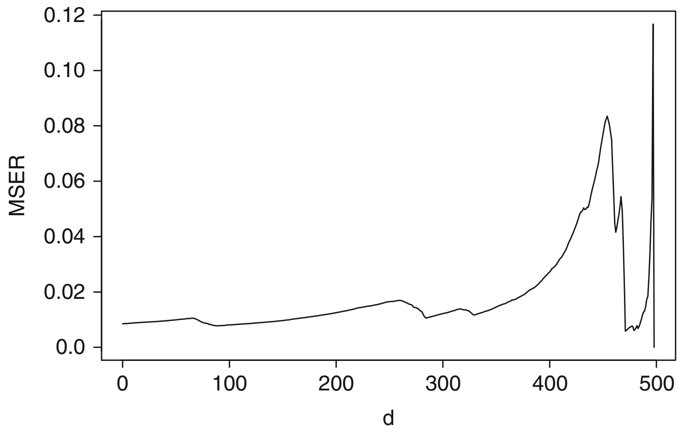

Fig. 8.3 The MSER(d) statistic for the $M/G/1$ example with run length $m = 500$ customers. In this example $\widehat{d} = \tilde{d} = 88$

Using this formula, the MSER statistic can be computed in one pass through the data by starting from the end of the series and working backward to the beginning:

```
1. Set $s = 0, q = 0$
2. For $d = m - 1$ to 0
    a. $s = s + Y_{d+1}$
    b. $q = q + Y_{d+1}^2$
    c. $\mathrm{MSER}(d) = \left(q - s^2/(m-d)\right)/(m-d)^2$
3. Next $d$
```

Figure 8.3 shows a plot of MSER(d) for one replication of $m = 500$ waiting times from the $M/G/1$ simulation of Sect. 4.3. In this case $\widehat{d} = \tilde{d} = 88$, which suggests that estimating the steady-state waiting time by averaging the last $500 - 88 = 412$ values will minimize the MSE of the estimate.

8.2.2.2 Error Estimation with a Fixed Budget

When we had n replications, we exploited the fact that

$$\mathrm{Var}\left(\overline{Y}(n,m,d)\right) = \frac{\mathrm{Var}\left(\overline{Y}(1,m,d)\right)}{n}$$

because replications allowed us to estimate $\mathrm{Var}\left(\overline{Y}(1,m,d)\right)$. The relationship between the variance of the sample mean and the variance of the component replications is one of the most important in all of statistics.

When we only have a single replication, $Y_1, Y_2, \ldots, Y_m$ (perhaps after some deletion), we can exploit a similar relationship, provided m is sufficiently large: Let $\overline{Y}(m)$ be the overall sample mean (dropping d and n to simplify notation). Then we know that $\mathrm{Var}\left(\overline{Y}(m)\right) \approx \gamma^2/m$. Here is the key insight: For $b < m$, but still large, $\mathrm{Var}\left(\overline{Y}(b)\right) \approx \gamma^2/b$ as well. Therefore, if both m and b are large,

$$\mathrm{Var}\left(\overline{Y}(m)\right) \approx \frac{b}{m}\mathrm{Var}(\overline{Y}(b)). \tag{8.16}$$

However, rather than making replications, we can estimate $\mathrm{Var}\left(\overline{Y}(b)\right)$ by forming $k = m/b$ batch means as shown below:

$$\underbrace{Y, \ldots, Y}_{\text{deleted}}, \underbrace{Y_1, \ldots, Y_b}_{\overline{Y}_1(b)}, \underbrace{Y_{b+1}, \ldots, Y_{2b}}_{\overline{Y}_2(b)}, \ldots, \underbrace{Y_{(k-1)b+1}, \ldots, Y_{kb}}_{\overline{Y}_k(b)}.$$

We refer to b as the batch size, and k as the number of batches. The natural estimator of $\mathrm{Var}\left(\overline{Y}(b)\right)$ is then the sample variance of the batch means

$$S^2(k) = \frac{1}{k-1}\sum_{j=1}^{k}\left(\overline{Y}_j(b) - \overline{Y}(m)\right)^2.$$

This yields the batch means variance estimator $\widehat{\mathrm{Var}}\left(\overline{Y}(m)\right) = (b/m)S^2(k)$, or equivalently the batch means estimator of the asymptotic variance constant $\widehat{\gamma}^2 = bS^2(k)$. Finally, an approximate confidence interval for μ is

$$\overline{Y}(m) \pm t_{1-\alpha/2,k-1}\frac{\widehat{\gamma}}{\sqrt{m}}$$

which is algebraically equivalent to

$$\overline{Y}(m) \pm t_{1-\alpha/2,k-1}\frac{S(k)}{\sqrt{k}}. \tag{8.17}$$

The development above made use of a lot of "$\approx$" arguments. When might this be a good approximation? Recall from Eq. (5.7) that

$$\gamma^2 = \sigma^2\left(1 + 2\sum_{i=1}^{\infty}\rho_i\right).$$

Because the correlations have to be summable for γ^2 to exist, we expect the autocorrelations to diminish with i, and to be effectively 0 for $i > b^\star$, for some $b^\star$. Therefore, $\sigma^2\left(1 + 2\sum_{i=1}^{b^\star}\rho_i\right)$ is a really good approximation of γ^2. If the batch size $b \geq b^\star$ (which of course means $m > b^\star$ as well), then the relationship (8.16) holds. Thus, the batch size needs to capture the correlation structure of the output process.

For the confidence interval (8.17) to be asymptotically valid requires a bit more: As the sample size m goes to infinity with the number of batches k fixed (so that the batch size is also increasing), the batch means $\overline{Y}_1(b), \overline{Y}_2(b), \ldots, \overline{Y}_k(b)$ need to become independent and normally distributed, which can be shown to be true under relatively mild conditions (e.g., Steiger and Wilson, 2001).

The problem of deciding when b is large enough is a difficult one. Obviously the larger b is the more likely (8.16) is to hold. A number of data-driven algorithms for selecting batch size have been published, with the goal of either delivering a valid confidence interval or standard error estimate at the end. Some algorithms also control the run length to achieve a fixed precision, while others assume the run length is given. An example, with references to the broader literature, is Tafazzoli and Wilson (2011).

The data-driven batching algorithms are sometimes complicated, meaning that implementation could involve substantial effort. If the simulation budget N is really fixed and cannot be extended, then the following procedure is sensible and relatively simple to apply:

1. Make a single replication of length $m = N$.
2. Apply MSER to obtain a deletion point d (or for an even simpler procedure, set $d = 0$).
3. Divide the remaining $m - d$ observations into from $10 \leq k \leq 30$ batches, looking for a value of k that divides $m - d$ close to evenly (if there are data left over, delete from the beginning).
4. Compute the sample mean and form the batch means confidence interval (8.17).

The assumption behind this approach is that $m = N$ is very large, large enough that it is possible to estimate μ precisely and form an approximately valid confidence interval (and possibly even large enough that data deletion is not necessary). When this is the case, the analysis in Schmeiser (1982) shows that little is lost by using a relatively small number of batches $10 \leq k \leq 30$, while on the other hand there is risk if the number of batches k is too large making the batch size $b < b^\star$ (too small).

The reason that little is lost provided $k \geq 10$ can be understood by observing the impact of number of batches on (8.17). Notice that the confidence interval depends on the estimator $\widehat{\gamma}^2$ of the asymptotic variance constant, γ^2, and the batch means variance estimator typically will be valid for any value of k small enough that $b > b^\star$.

So why not pick $k = 2$? Consider the ratio of $t_{1-\alpha/2,k-1}$ to $z_{1-\alpha/2} = t_{1-\alpha/2,\infty}$ for $1 - \alpha = 0.95$, shown in Table 8.1. At $k = 2$ it is about 6.5 times larger than it would be if k was very large. However, at $k = 10$ it is only 1.15 times larger, dropping to 1.04 times larger at $k = 30$ batches. Since the risk of an invalid confidence interval increases as k increases, $10 \leq k \leq 30$ is a good working range.

The method described here for estimating γ^2 is usually called the *nonoverlapping batch means method*, and it is one of many that have been proposed. An overview can be found in Goldsman and Nelson (2006).

For the $M/G/1$ simulation of Sect. 4.3 we made a single replication of $m =$ 30,000 customer waiting times. The minimum MSER(d) occurred at $\widehat{d} = \tilde{d} = 0$,

Table 8.1 Relative impact of number of batches k on the 0.975 quantile of the t distribution

k	$\frac{t_{0.975,k-1}}{z_{0.975}}$
2	6.48
5	1.42
10	1.15
20	1.07
30	1.04
60	1.02
∞	1.00

which is not surprising with a long run for a system that exhibits little bias: apparently the benefit from reducing the (small) bias with 30,000 observations was not substantial relative to the increased variance from deleting data. These data were then batched into $k = 30$ batches of size $b = 1000$ waiting times and the sample means computed from each batch. Applying the CI (8.17) yielded the 95% CI 2.24 ± 0.22 min.

What if an error of ± 0.22 min is considered too large? Ideally, the run length m should be increased and the data are rebatched into $10 \le k \le 30$ batches. If the raw data were not retained, then additional batches of the same size b should be generated, increasing k.

8.2.3 Batch Statistics

The preceding sections covered estimating the mean of a steady-state simulation output process from discrete-time output data. However, performance measures such as queue length and inventory level are continuous-time outputs, $Y(t), 0 \le t \le T$. In the fixed-budget, single-replication setting, continuous-time outputs can be converted into discrete-time outputs using the batching transformation

$$Y_i = \frac{1}{\Delta t}\int_{(i-1)\Delta t}^{i\Delta t} Y(t)\,\mathrm{d}t \tag{8.18}$$

for $i = 1, 2, \ldots, m$ where $m = \lfloor T/\Delta t \rfloor$ and $\Delta t > 0$ is a time interval. For instance, if $Y(t)$ is queue length, time $t = 0$ corresponds to 8 a.m., and $\Delta t = 15$ min, then Y_7 is the average queue length from 9:30 a.m. (time 90 min) to 9:45 a.m. (time 105 min). Tools such as MSER and batch means can then be applied to $Y_1, Y_2, \ldots, Y_m$. It is important that Δt be large enough that the value of $Y(t)$ is likely to change one or more times during the interval; otherwise the transformed process $Y_1, Y_2, \ldots, Y_m$ may exhibit extremely strong positive correlation with the same value appearing two or more times in succession.

As shown at the end of Sect. 8.1 for across-replication data, the concept of batching can be applied to performance measures other than means and this is the case even for a single, long replication of a steady-state simulation provided the performance measure θ can be consistently estimated by a statistic $\widehat{\theta}$ that is a function of stationary, but dependent data. The basic outputs from the replication are divided into k batches of b individual observations, and from the jth batch we compute a $\widehat{\theta}_j(b)$ statistic:

$$\underbrace{Y,\ldots,Y}_{\text{deleted}},\underbrace{Y_1,\ldots,Y_b}_{\widehat{\theta}_1(b)},\underbrace{Y_{b+1},\ldots,Y_{2b}}_{\widehat{\theta}_2(b)},\ldots,\underbrace{Y_{(k-1)b+1},\ldots,Y_{kb}}_{\widehat{\theta}_k(b)}.$$

The performance measure θ is estimated by the average

$$\overline{\theta}(k) = \frac{1}{k}\sum_{j=1}^{k}\widehat{\theta}_j(b)$$

and not by applying $\widehat{\theta}$ to the entire data set; in the case of batch means the average of the batch means is equal to the overall average, but this is not true for other statistics.

For example, suppose that we want to estimate the marginal variance $\sigma^2 = \mathrm{Var}(Y)$. If $Y_1, Y_2, \ldots, Y_m$ is a stationary but possibly dependent output process (say, after deletion) with $\mathrm{Var}(Y_i) = \sigma^2$, then it can be shown under mild conditions that

$$\frac{1}{m-1}\sum_{i=1}^{m}\left(Y_i - \overline{Y}(m)\right)^2 \xrightarrow{a.s.} \sigma^2$$

as $m \to \infty$. That is, it is a consistent estimator even though the output data are dependent. To apply the batching method, the batch statistics are

$$\widehat{\theta}_j(b) = \frac{1}{b-1}\sum_{i=(j-1)b+1}^{jb}\left(Y_i - \overline{Y}_j(b)\right)^2,\ j = 1,2,\ldots,k$$

the sample variance from within each batch of observations. The marginal variance σ^2 is estimated by

$$\overline{\theta}(k) = \frac{1}{k}\sum_{j=1}^{k}\widehat{\theta}_j(b)$$

the average of the batch statistics, and a confidence interval is formed as in (8.17) using the variance estimator

$$S^2(k) = \frac{1}{k-1}\sum_{j=1}^{k}\left(\widehat{\theta}_j(b) - \overline{\theta}(k)\right)^2.$$

8.2.4 Steady-State Simulation: A Postscript

Two things about the steady-state simulation problem can be stated with confidence: it is a difficult problem to solve, and there have been many, many attempts to solve it. The tools presented in the previous sections were selected because they are easy to use and they illustrate what we believe are the important concepts. There are, however, other methods that may have significant advantages and new methods are being invented regularly. The best way to research solutions past and present is via the archive of the *Proceedings of the Winter Simulation Conference* at http://www.wintersim.org/.

In any event, the important concepts are these:

1. There is a tradeoff between bias and variance, and both matter, as reflected in the MSE.
2. Bias diminishes faster than variance with increasing run length, but does not diminish at all with increasing numbers of replications. Thus, no solution will have a very large number of short replications.
3. How you attack the steady-state simulation problem depends on whether you can afford (usually in terms of time available) to treat it as a fixed-precision or a fixed-budget problem. When data are cheap (i.e., fast) a fixed-precision approach may waste data but it gives more assurance that you get what you want.

We suggested here that in the fixed-budget case statistical analysis favors one long run ($n = 1$ replication of length $m = N$). However, the increasing availability of cheap, parallel computing makes single-replication strategies less likely to be employed in the future. For instance, if your personal computer has p cores, then you can make $n = p$ replications of length $m = N$ in roughly the same time it takes to make one. So why would you ever make just one? When one has to purchase parallel computing from a service, then of course there is a cost to increasing p, but that cost is declining rapidly.

8.3 Variance Reduction Using Control Variates

In this final section we consider the choice of estimator $\widehat{\theta}(\mathbf{x};T,n,\mathbf{U})$ as part of experiment design. While sample means are often appropriate, sometimes we can do better.

We have emphasized measuring and controlling estimation error through the number of replications or run length, and for many simulations that approach is sufficient. But estimation error typically declines as $1/\sqrt{\text{effort}}$, where "effort" is measured in replications or run length or both. Thus, the decline is slow and there are problems for which the simulation experiment is too computationally expensive to attain the level of error we want with the amount of simulation effort we can afford or have time to spend. Examples include:

- Estimating the performance of thousands of alternatives so that it is not possible to spend substantial simulation effort on each of them.

- Estimating performance measures associated with rare events so that very long simulations are required to observe even one occurrence of the event.
- Estimating performance parameters for which very little error can be tolerated when the outputs themselves are highly variable.

Variance reduction techniques (VRTs) are strategies for reducing estimation error without a corresponding increase in simulation effort. Common random numbers, described in Chap. 9, is a VRT that reduces the variance of the estimated difference between two systems' performance measures relative to using independent simulations. "Variance reduction" is always relative to what the variance would have been had we not used the VRT.

VRTs often have to be carefully tailored to the specific simulation of interest. Good references are Bratley et al. (1987), Glasserman (2004), Asmussen and Glynn (2007) and Law (2007). Here we describe a VRT called *control variates* that, like common random numbers, is widely applicable. In Sect. 9.11 we consider another VRT called *importance sampling*.

Control variates apply when the goal is to estimate $\mu_Y = \mathrm{E}(Y)$ for some simulation output Y; it can be motivated by a fact about variance: Let C be some other random variable that is also observable in the simulation. Then

$$\mu_Y = \mathrm{E}(Y) = \mathrm{E}[\mathrm{E}(Y|C)] \tag{8.19}$$

$$\sigma_Y^2 = \mathrm{Var}(Y) = \mathrm{Var}[\mathrm{E}(Y|C)] + \mathrm{E}[\mathrm{Var}(Y|C)]. \tag{8.20}$$

Recall that $\mathrm{E}(Y|C)$, the conditional expected value of output Y given C, is itself a random variable. In the double-expectation result (8.19) and the variance-decomposition result (8.20), the inner expectation is with respect to the conditional distribution of Y given C, while the outer is with respect to the distribution of C. Together these results imply that the random variable $\mathrm{E}(Y|C)$ is an unbiased estimator of μ_Y, it has no larger variance than Y, and it likely has smaller variance since we expect $\mathrm{E}[\mathrm{Var}(Y|C)] > 0$. Intuitively, the stronger the association between Y and C, the greater the variance reduction. Using $\mathrm{E}(Y|C)$ instead of Y to estimate μ_Y is a VRT. However, in most real problems we will not know a conditional expected value $\mathrm{E}(Y|C)$ that we can exploit, so we use these results in a different way.

The equalities (8.19) and (8.20) imply that we can represent the simulation output Y as follows (where we have listed the variance contribution below each term)

$$\underbrace{Y}_{\sigma_Y^2} = \underbrace{\mathrm{E}(Y|C)}_{\mathrm{Var}[\mathrm{E}(Y|C)]} + \underbrace{\varepsilon}_{\mathrm{E}[\mathrm{Var}(Y|C)]},$$

where $\varepsilon = Y - \mathrm{E}(Y|C)$, so that it has expected value 0, variance $\mathrm{E}[\mathrm{Var}(Y|C)]$ and ε is independent of $\mathrm{E}(Y|C)$ (independence is not obvious but can be shown). Because we observe Y and C we have some hope of estimating the relationship $\mathrm{E}(Y|C)$ and therefore (mostly) removing it as a contributor to variance. Control variates does this by using a linear approximation

$$\mathrm{E}(Y|C) = \beta_0 + \beta_1(C - \mu_C), \tag{8.21}$$

where $\mu_C = \mathrm{E}(C)$ and is *known*, and β_0 and β_1 are constants. Then it follows immediately that $\beta_0 = \mu_Y$, since

$$\mu_Y = \mathrm{E}(Y) = \mathrm{E}\left[\mathrm{E}(Y|C)\right] = \mathrm{E}[\beta_0 + \beta_1(C - \mu_C)] = \beta_0.$$

This gives the control-variate model

$$Y = \mu_Y + \beta_1(C - \mu_C) + \varepsilon. \tag{8.22}$$

Is the representation (8.22) even plausible? Here is one justification: If the joint distribution of (Y,C) is bivariate normal,

$$\begin{pmatrix} Y \\ C \end{pmatrix} \sim \mathrm{BVN}\left(\begin{bmatrix} \mu_Y \\ \mu_C \end{bmatrix}, \begin{bmatrix} \sigma_Y^2 & \rho\,\sigma_Y\sigma_C \\ \rho\,\sigma_Y\sigma_C & \sigma_C^2 \end{bmatrix} \right) \tag{8.23}$$

where ρ is the correlation between Y and C, then it is known that

$$\mathrm{E}(Y|C) = \mu_Y + \beta_1^\star(C - \mu_C),$$

where

$$\beta_1^\star = \frac{\mathrm{Cov}(Y,C)}{\mathrm{Var}(C)} = \frac{\rho\,\sigma_Y\sigma_C}{\sigma_C^2}$$

(see, e.g., Kotz et al., 2000). If we remove all of the variance due to $\mathrm{E}(Y|C)$, then the leftover variance is $\mathrm{E}[\mathrm{Var}(Y|C)] = (1-\rho^2)\sigma_Y^2 \le \sigma_Y^2 = \mathrm{Var}(Y)$, with the variance decreasing the larger the value of the squared correlation between Y and C.

Of course, we cannot count on our outputs being jointly normal. But in a simulation experiment we observe $(Y_i, C_i), i = 1,2,\ldots,n$, across n replications and base our analysis on $(\overline{Y}, \overline{C})$. Then a multivariate version of the central limit theorem implies that under mild conditions the sample means $(\overline{Y}, \overline{C})$ will tend to be approximately bivariate normal if n is large enough. This provides one justification for the approximation (8.22).

When we have replications, the form of (8.22) suggests estimating $\beta_0 = \mu_Y$ via least-squares regression. The regression formulation is

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix} = \begin{pmatrix} 1 & C_1 - \mu_C \\ 1 & C_2 - \mu_C \\ \vdots & \vdots \\ 1 & C_n - \mu_C \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} = \mathbf{C}\beta + \varepsilon.$$

This gives the least-squares estimators

$$\begin{pmatrix} \widehat{\beta}_0 \\ \widehat{\beta}_1 \end{pmatrix} = \widehat{\beta} = (\mathbf{C}^\top\mathbf{C})^{-1}\mathbf{C}^\top\mathbf{Y}$$

of which $\widehat{\beta}_0$ is the first element. In more detail,

$$\widehat{\beta}_0 = \overline{Y} - \widehat{\beta}_1(\overline{C} - \mu_C), \tag{8.24}$$

where

$$\widehat{\beta}_1 = \frac{\sum_{i=1}^n (Y_i - \overline{Y})(C_i - \overline{C})}{\sum_{i=1}^n (C_i - \overline{C})^2}. \tag{8.25}$$

We refer to $\widehat{\beta}_0$ as a *control-variate estimator* of μ_Y and C as a *control*. Expression (8.24) provides additional intuition for this approach: The term $\widehat{\beta}_1(\overline{C} - \mu_C)$ adjusts the usual estimator $\overline{Y}$ based on the difference between $\overline{C}$ and its expected value μ_C.

Does $\widehat{\beta}_0$ have smaller variance than $\overline{Y}$? Under very mild conditions $\widehat{\beta}_0$ satisfies the central limit theorem

$$\sqrt{n}(\widehat{\beta}_0 - \mu_Y) \xrightarrow{D} \mathrm{N}\left(0, (1-\rho^2)\sigma_Y^2\right) \tag{8.26}$$

as $n \to \infty$ (Nelson, 1990). Thus, for large n

$$\mathrm{Var}\left(\widehat{\beta}_0\right) \approx (1-\rho^2)\frac{\sigma_Y^2}{n} \le \frac{\sigma_Y^2}{n} = \mathrm{Var}(\overline{Y}).$$

This shows that if we use the regression approach, then the bivariate normal result holds in the limit even when (Y, C) are not bivariate normally distributed and β_1 must be estimated.

To provide a measure of estimation error on $\widehat{\beta}_0$ we can use the regression estimator of the variance–covariance matrix of $\widehat{\beta}$:

$$\widehat{\Sigma} = \widehat{\mathrm{Var}}(\widehat{\beta}) = (\mathbf{Y}^\top\mathbf{Y} - \widehat{\beta}^\top\mathbf{C}^\top\mathbf{Y})(\mathbf{C}^\top\mathbf{C})^{-1}/(n-2).$$

The $(1,1)$ element is the estimated $\mathrm{Var}(\widehat{\beta}_0)$, and an approximate confidence interval is

$$\widehat{\beta}_0 \pm t_{1-\alpha/2, n-2}\sqrt{\widehat{\Sigma}_{11}}.$$

Since $\widehat{\Sigma}_{11}$ is the variance estimator for the intercept β_0 in regression it is generated by standard regression software. In detail, it is

$$\widehat{\Sigma}_{11} = \frac{\sum_{j=1}^n \left(Y_j - \widehat{\beta}_0 - \widehat{\beta}_1(C_j - \mu_C)\right)^2}{n-2}\left(\frac{1}{n} + \frac{(\overline{C} - \mu_C)^2}{\sum_{j=1}^n (C_j - \overline{C})^2}\right).$$

This development makes clear that when we are looking for a control C two things are essential: The expected value of C must be known, and C should be strongly correlated with Y. Some possibilities from our standard examples follow:

8.3.1 M/G/1 Queue Control Variate

Recall that in the $M/G/1$ queueing simulation of Sect. 4.3 the output is the average of the last $m-d$ customer waiting times,

$$\overline{Y} = \frac{1}{m-d} \sum_{i=d+1}^{m} Y_i,$$

where

$$\begin{aligned} Y_0 &= 0 \qquad X_0 = 0 \\ Y_i &= \max\{0, Y_{i-1} + X_{i-1} - A_i\},\ i = 1,2,\ldots,m \end{aligned}$$

and we are interested in the steady-state expected waiting time $\mu = \mathrm{E}(Y)$. One possible control is the average difference between the service time of customer $i-1$ and the interarrival time between customer $i-1$ and customer i:

$$\overline{C} = \frac{1}{m-d} \sum_{i=d+1}^{m} (X_{i-1} - A_i).$$

For the parameter settings in Sect. 4.3 $\mu_C = 1 - 0.8 = -0.2$.

When we make n replications we observe a pair of averages $(\overline{Y}_j, \overline{C}_j)$ on each one. Therefore, the regression set up is

$$\begin{pmatrix} \overline{Y}_1 \\ \overline{Y}_2 \\ \vdots \\ \overline{Y}_n \end{pmatrix} = \begin{pmatrix} 1 & \overline{C}_1 - \mu_C \\ 1 & \overline{C}_2 - \mu_C \\ \vdots & \vdots \\ 1 & \overline{C}_n - \mu_C \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

Rerunning the example the control-variate estimator provides 95% confidence interval $[2.109, 2.187]$ as compared to $[2.101, 2.196]$ without it. The estimated correlation between the average waiting time $\overline{Y}$ and $\overline{C}$ is 0.648, a rather weak correlation which explains why there is little difference in the confidence-interval widths.

8.3.2 Stochastic Activity Network Control Variate

Recall that in the stochastic activity network of Sect. 4.4 the output is the time to complete the project

$$Y = \max\{X_1 + X_4, X_1 + X_3 + X_5, X_2 + X_5\}$$

and we are interested in $\theta = \Pr\{Y > t_p\}$. One possible control is $C = X_1 + X_3 + X_5$, the path with the longest expected value: Since X_1, X_3, and X_5 are independent, exponentially distributed random variables each with mean 1, C has an Erlang distribution with mean 3 and three phases. Using this fact we can show that

$$\theta_C = \Pr\{C > t_p\} = (1 + t_p + t_p^2/2)\mathrm{e}^{-t_p}$$

which is easy to compute. The regression set up in this case is

$$\begin{pmatrix} I(Y_1 > t_p) \\ I(Y_2 > t_p) \\ \vdots \\ I(Y_n > t_p) \end{pmatrix} = \begin{pmatrix} 1 & I(C_1 > t_p) - \theta_C \\ 1 & I(C_2 > t_p) - \theta_C \\ \vdots & \vdots \\ 1 & I(C_n > t_p) - \theta_C \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

Since the output and the control are indicator (0 or 1) random variables, they clearly do not have a bivariate normal distribution; nevertheless, the central limit theorem result implies that we can attain a variance reduction when we have a large number of replications n.

Rerunning the example at $t_p = 5$, the control-variate estimator provides a 95% confidence interval $[0.165, 0.194]$ as compared to $[0.140, 0.186]$ without it. In this experiment $n = 1000$. The estimated correlation between $I(Y > t_p)$ and $I(C > t_p)$ is 0.78, and there is a small, but noticeable reduction in confidence-interval width.

8.3.3 Asian Option Control Variate

This example is derived from Glasserman (2004, Chaps. 3 and 4).

Recall that in the Asian option example of Sect. 4.5 the goal was to estimate

$$\nu = \mathrm{E}\left[\mathrm{e}^{-rT}\left(\overline{X}(T) - K\right)^+\right]$$

using the output $Y = \mathrm{e}^{-rT}\max\left\{0, \widehat{\overline{X}(T)} - K\right\}$ where

$$\widehat{\overline{X}(T)} = \frac{1}{m}\sum_{i=1}^{m} X(i\Delta t).$$

A closely related problem is estimating

$$\nu_C = \mathrm{E}\left[\mathrm{e}^{-rT}\left(\tilde{X}(T) - K\right)^+\right],$$

where

$$\tilde{X}(T) = \left(\prod_{i=1}^{m} X(i\Delta t)\right)^{1/m}$$

is the geometric average of the asset values. The value of an Asian option based on the geometric average is analytically solvable, so $C = \mathrm{e}^{-rT}\max\{0, \tilde{X}(T) - K\}$ is a potential control. The formula for ν_C is given in the Exercises.

Rerunning the example the control-variate estimator provides a 95% confidence interval [\$2.17, \$2.18] as compared to [\$2.10, \$2.29] without it. The estimated correlation between Y and C is 0.999, which explains the dramatic reduction in the confidence-interval width.

Appendix: Properties of Control-Variate Estimators

This appendix provides some small-sample (finite n) properties of control-variate estimators. All of these results assume that the output and the control are linearly related; we add assumptions as needed to obtain properties. These assumptions are stronger than required for the central limit theorem result (8.26), but lead to results that apply for finite n. Complete details can be found in Nelson (1990).

1. Under model (8.21), $\mathrm{E}(\widehat{\beta}_1) = \beta_1$.

   Proof. For convenience, let $\mathbf{C}^\top = (C_1, C_2, \ldots, C_n)$. Then

$$\begin{aligned}
\mathrm{E}(\widehat{\beta}_1|\mathbf{C}=\mathbf{c}) &= \frac{\sum \mathrm{E}((Y_i - \overline{Y})|\mathbf{C}=\mathbf{c})(c_i - \overline{c})}{\sum (c_i - \overline{c})^2} \\
&= \frac{\sum(\mu_Y + \beta_1(c_i - \mu_C) - \mu_Y - \beta_1(\overline{c} - \mu_C))(c_i - \overline{c})}{\sum(c_i - \overline{c})^2} \\
&= \beta_1.
\end{aligned}$$

   So that, by the double expectation theorem, $\mathrm{E}(\widehat{\beta}_1) = \beta_1$. □

2. Under model (8.21), $\mathrm{E}(\widehat{\beta}_0) = \mu_Y$.

   Proof. $\mathrm{E}(\widehat{\beta}_0|\mathbf{C}=\mathbf{c}) = \mathrm{E}(\overline{Y}|\mathbf{C}=\mathbf{c}) - \mathrm{E}(\widehat{\beta}_1|\mathbf{C}=\mathbf{c})(\overline{c} - \mu_C) = \mu_Y + \beta_1(\overline{c} - \mu_C) - \beta_1(\overline{c} - \mu_C) = \mu_Y$ using the result above. Since this is independent of $\mathbf{c}$ the conclusion holds in general. □

   Thus, the control-variate estimator is unbiased under the assumed linear model. When will it have smaller variance than $\overline{Y}$? Notice that

$$\mathrm{Var}(\widehat{\beta}_0) = \mathrm{Var}\left[\mathrm{E}(\widehat{\beta}_0|\mathbf{C})\right] + \mathrm{E}\left[\mathrm{Var}(\widehat{\beta}_0|\mathbf{C})\right] = \mathrm{E}\left[\mathrm{Var}(\widehat{\beta}_0|\mathbf{C})\right]$$

   since $\mathrm{Var}\left[\mathrm{E}(\widehat{\beta}_0|\mathbf{C})\right] = \mathrm{Var}[\mu_Y] = 0$ from the proof of the result above. Still, this expression is not easy to evaluate in general. Under the additional assumption of constant conditional variance ($\mathrm{Var}(Y|C) = \sigma^2$), the following result can be (tediously) derived:

3. Under model (8.21) and the assumption of constant conditional variance,

$$\mathrm{Var}(\widehat{\beta}_0|\mathbf{C}) = \sigma^2\left(\frac{1}{n} + \frac{(\overline{C}-\mu_C)^2}{\sum(C_i-\overline{C})^2}\right).$$

To complete the calculation we need to take the expectation of this result, which depends on the distribution of C. If (Y,C) are jointly normally distributed with correlation ρ, then we know that $\sigma^2 = (1-\rho^2)\sigma_Y^2$ and it can be shown that

$$\mathrm{E}\left(\frac{(\overline{C}-\mu_C)^2}{\sum(C_i-\overline{C})^2}\right) = \frac{1}{n(n-3)}.$$

This leads to the following:

4. If (Y,C) are bivariate normal, then

$$\mathrm{Var}(\widehat{\beta}_0) = \left(\frac{n-2}{n-3}\right)(1-\rho^2)\frac{\sigma_Y^2}{n}.$$

Therefore, if $\rho^2 > 1/(n-2)$, then the control-variate estimator has smaller variance than the sample mean. Thus, provided the number of replications is not too small the control-variate estimator will have smaller variance even if the correlation is weak.

Exercises

1. Verify (8.6) and (8.7).
2. Show that for the AR(1) process the asymptotic MSE is

$$\mathrm{MSE}\left(\overline{Y}(n,m,d)\right) \approx \frac{(y_0-\mu)^2\varphi^{2d+2}}{(m-d)^2(1-\varphi)^2} + \frac{\sigma^2}{n(m-d)(1-\varphi)^2}.$$

Then prove that for $\varphi > 0$ the asymptotic squared bias $(y_0-\mu)^2\varphi^{2d+2}/(m-d)^2(1-\varphi)^2$ is decreasing in d for m large enough. *Hint*: Take the derivative with respect to d.

3. Use the result in Exercise 2 to study the impact of dependence (as measured by φ) and initial conditions (as measured by $(y_0-\mu)^2$) on the optimal deletion point d. A similar study was undertaken by Snell and Schruben (1985).
4. Using the result from Exercise 2, show that for the AR(1) process and m large

$$\mathrm{E}\left(\mathrm{MSER}(d)\right) \approx \frac{(1-\varphi)^2}{(1-\varphi^2)}\left(\frac{(y_0-\mu)^2\varphi^{2d+2}}{(m-d)^2(1-\varphi)^2} + \frac{\sigma^2}{n(m-d)(1-\varphi)^2}\right).$$

That is, the expected value of MSER is proportional to the asymptotic MSE.

5. As an aid to computing the MSER statistic, show that

$$\sum_{i=d+1}^{m} \left(Y_i - \overline{Y}(m,d)\right)^2 = \sum_{i=d+1}^{m} Y_i^2 - \frac{1}{m-d}\left(\sum_{i=d+1}^{m} Y_i\right)^2 .$$

6. For a covariance stationary process $Y_1, Y_2, \ldots, Y_m$ we know that the variance of the sample mean is

$$\mathrm{Var}(\overline{Y}) = \frac{\sigma^2}{m}\left(1 + 2\sum_{k=1}^{m-1}\left(1 - \frac{k}{m}\right)\rho_k\right),$$

where σ^2 is the marginal variance and ρ_k is the lag-k autocorrelation. If we have data, it would seem that we could estimate each term in this expression directly. What problems can you identify with this direct approach?

7. Remote Order Taking (ROT) is proposing to replace the current drive-through window ordering for a fast-food restaurant chain with the equivalent of a call center. ROT promises lower cost and faster response. Before agreeing to this, the corporate owner of the restaurant chain has asked for a proof-of-concept study using data from seven of their stores in Columbus, Indiana.

   The typical drive-through window has a single traffic lane featuring a menu-board/order-entry station, a window where the person taking orders also collects payment, and a second window where the food is delivered to the customer. ROT is proposing a service that will eliminate the need for the first window and the corresponding employee by providing high-speed voice and data connections that will allow operators in Rapid City, South Dakota, to take orders and communicate them to the restaurant. The second window will then be used to collect payment and deliver the food to the customer.

   This restaurant chain has high standards for customer service. When asked by ROT, they quantified this by saying that they want the average response time from when a drive-through customer's car triggers the sensor at the order board until that customer is greeted by the order taker to be 3 s or less, and the chance that any customer's response time is greater than 7 s be no more than 20%.

   Whether or not ROT's proposal is worthwhile depends on whether the number of operators required to achieve these two goals is significantly less than one per store. The restaurant has provided data on the busiest 3-h period from the seven stores in the Columbus area to allow ROT to do a proof-of-concept study. You have been hired by ROT to simulate serving these seven Columbus stores to determine the minimum number of operators required to meet the service-level requirements.

   Modeling the arrival process to each store as Poisson seems reasonable. From a prior study, data on the time required to take an order and the time for a car to clear the order board and the next car in line (if there is one) to pull up has been collected; that data can be found on the book website in `ROTData.xls`.

Store number	Average number of customers (11 a.m. to 2 p.m.)
1	36
2	15
3	45
4	26
5	36
6	36
7	45

Clearly, arrival rates will vary by time of day. Since we only have data on the busiest period, treat this as a steady-state simulation where the long-run arrival rate is the same as the busy period; staffing that can keep up over the long-run at the heaviest load will certainly be adequate generally. Therefore, you will need to deal with initial-condition bias.

Notice that the entire food-preparation and food-delivery function is outside the scope of this project. You are only interested in order taking, and your goal is to investigate how many operators would be required to serve the seven stores.

8. Suppose that $Y_1, Y_2, \ldots, Y_m$ is a covariance stationary process with mean μ. Let

$$\widehat{\sigma}^2 = \frac{1}{m}\sum_{i=1}^{m}(Y_i-\mu)^2$$

be the sample variance when the mean μ is known. Derive an expression for $\mathrm{E}(\widehat{\sigma}^2/m)$ and compare it to the expression in Exercise 6, the true variance of the sample mean. Is $\widehat{\sigma}^2/m$ an unbiased estimator of $\mathrm{Var}\left(\overline{Y}(m)\right)$? Next do the same analysis for

$$S^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(Y_i-\overline{Y}(m)\right)^2$$

the usual sample variance and compare $\mathrm{E}(S^2/m)$ to the expression in Exercise 6.

9. When we only have a single replication $Y_1, Y_2, Y_3, \ldots, Y_m$ of a steady-state simulation, an alternative to the MSER statistic for determining a deletion point is to plot the *cumulative average*

$$\overline{Y}(j) = \frac{1}{j}\sum_{i=1}^{j} Y_i$$

for $j = 1, 2, 3, \ldots, m$, and truncate at the point when this plot becomes flat. To examine this idea, assume that the underlying output is an AR(1) process

$$Y_{i+1} = \mu + \varphi(Y_i-\mu) + X_{i+1}$$

with $y_0 \neq \mu$, $\varphi > 0$ and $X_1, X_2, \ldots$ i.i.d. with mean 0 and variance σ^2. Provide analysis that shows why this approach is conservative, meaning that it will tend to give a larger deletion point than necessary.

10. Recall the MA(1) surrogate process of Exercise 11 in Chap. 5:

$$Y_i = \mu + \theta X_{i-1} + X_i$$

with $X_0 = x_0$ a fixed constant, and $X_1, X_2, \ldots$ i.i.d. with mean 0 and variance σ^2, and $|\theta| < 1$. That exercise asked you to derive the asymptotic MSE of the sample mean $\overline{Y}(m) = \sum_{i=1}^{m} Y_i/m$ with no data deletion. Extend your result to obtain the asymptotic MSE for $d \geq 1$. If m is fixed, could it ever be optimal (minimize asymptotic MSE) to have $d > 0$, and if so what is the optimal d?
11. Recall the MA(1) surrogate process of Exercise 11 in Chap. 5:

$$Y_i = \mu + \theta X_{i-1} + X_i$$

with $X_0 = x_0$ a fixed constant, $X_1, X_2, \ldots$ i.i.d. with mean 0 and variance σ^2, and $|\theta| < 1$. For output data from an MA(1) process, compute the expected value of the MSER(d) statistic as a function of $d = 0, 1, 2, \ldots, m-1$. After how many observations do we expect deletion?
12. In Exercises 2 and 10 you derived the asymptotic MSE for AR(1) and MA(1) surrogate processes. Using these expressions you can approximate the MSE as a function of the initial conditions, run length m, deletion point d and values for the process parameters (φ, θ, etc.). Perform an empirical study to see how effective the minimum MSER statistic is at minimizing the MSE. In other words, simulate various cases of the AR(1) and MA(1) processes, estimate a deletion point using the minimum MSER rule, and evaluate how good the chosen deletion point is using the known asymptotic MSE.
13. Recall the $M/G/1$ simulation of Sect. 4.3, and specifically the steady-state waiting time Y. Using a single replication and batching, estimate and provide a 90% confidence interval for the marginal variance and the 0.8 quantile of Y.
14. If you could initialize a steady-state simulation in “steady-state conditions,” then there would be no initial-condition bias. Since this can rarely be done, it has been suggested that two replications of the simulation could be made such that the first is initialized in overloaded conditions, while the second is initialized in underloaded conditions, and then the two replications could be averaged to reduce the bias. For instance, the first replication of a queueing simulation could be initialized with many customers in the system, while the second replication could be initialized empty and idle. Assuming that the underlying output is an AR(1) process

$$Y_{i+1} = \mu + \varphi(Y_i - \mu) + X_{i+1}$$

with y_0 being a constant representing the initial conditions and X_i i.i.d. $(0, \sigma^2)$, show mathematically whether or not this idea always reduces bias relative to picking one fixed initial condition.
15. Work out what the variance-decomposition result (8.20) implies when (a) Y and C are independent, and when (b) $Y = C$.

16. Show that
$$\beta_1^\star = \frac{\mathrm{Cov}(Y,C)}{\mathrm{Var}(C)} = \frac{\rho\sigma_Y\sigma_C}{\sigma_C^2}$$
minimizes the variance of the random variable $\mu_Y + \beta_1^\star(C - \mu_C)$. This provides another justification for the control-variate estimator.
17. For the SAN example, use a control-variate estimator to estimate $\mu_Y = \mathrm{E}(Y)$, the expected time to complete the project. Compare it to just using $\overline{Y}$.
18. The control-variate estimator can employ more than one control, analogous to using more than one explanatory variable in linear regression. Try this idea on the $M/G/1$ simulation forming one control-variate based on the interarrival times, and another based on the service times.
19. The control-variate estimator can employ more than one control, analogous to using more than one explanatory variable in linear regression. Try this idea on the SAN simulation forming controls for each of the three paths through the network.
20. Specializing results in Glasserman (2004, Chap. 3) to our Asian option example (where the value of the asset is recorded at m equally spaced time intervals of length Δt), it can be shown that
$$v_C = \mathrm{e}^{-\delta T'} X(0)\Phi(d) - \mathrm{e}^{-rT'} K\Phi(d - \overline{\sigma}\sqrt{T'}),$$
where $T' = (m+1)\Delta t/2$, $\overline{\sigma}^2 = (2m+1)\sigma^2/(3m)$, $\delta = (\sigma^2 - \overline{\sigma}^2)/2$ and
$$d = \frac{\ln(X(0)/K) + (r - \delta + \overline{\sigma}^2/2)T'}{\overline{\sigma}\sqrt{T'}}.$$
Use this result to implement the control-variate estimator for the Asian call option. Test the effect of different numbers of discretization steps $m = 8, 16, 32, 64, 128$.

Chapter 9
Simulation Optimization and Sensitivity

We now focus on the problem of finding a good, or perhaps even the best, scenario $\mathbf{x}$ from a collection of feasible scenarios; when we are optimizing, the terms "solutions" and "systems" are synonyms for "scenario." We will primarily consider the formulation

$$\begin{aligned} &\min \ \theta(\mathbf{x}) \\ &\mathbf{x} \in \mathsf{C}, \end{aligned} \tag{9.1}$$

where $\theta(\mathbf{x})$ is the performance measure of interest, and C is the feasible set or region for the d-dimensional scenario variable $\mathbf{x}$. The feasible region C is often defined by a collection of constraints on $\mathbf{x}$, but it may also be just a list of options. What makes this a *simulation optimization* (SO) problem is the need to estimate the scenario performance measure $\theta(\mathbf{x})$ using a simulation-based estimator $\widehat{\theta}(\mathbf{x}; T, n, \mathbf{U})$. In Sect. 9.8 we discuss the situation in which satisfaction of the constraint $\mathbf{x} \in \mathsf{C}$ also has to be estimated, but for now we assume feasibility can be established without error. If the problem is more naturally formulated as a maximization, then stating it as $\min -\theta(\mathbf{x})$ converts it into the formulation (9.1).

Let $\mathbf{x}^\star$ denote the optimal scenario, which we initially assume is unique. Recall that our performance estimator is denoted $\widehat{\theta}(\mathbf{x}; T, n, \mathbf{U})$, where $\mathbf{x}$ is the scenario, T is the stopping time (run length), n is the number of replications, and $\mathbf{U}$ are the assigned pseudorandom numbers. To simplify notation let $\widehat{\theta}(\mathbf{x}) = \widehat{\theta}(\mathbf{x}; T, n, \mathbf{U})$ when we do not need to explicitly consider the other experiment design components T, n, and $\mathbf{U}$.

How we attack (9.1) depends on a number of things, including the type of feasible region C (discrete or continuous, for instance), the number of feasible scenarios in C (finite, countably infinite, uncountably infinite), and what we know about $\theta(\mathbf{x})$ and $\widehat{\theta}(\mathbf{x})$. We can usually expect that $\widehat{\theta}(\mathbf{x})$ is an unbiased, or at least a consistent,

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_9

estimator of $\theta(\mathbf{x})$. And the SO algorithm will simulate K scenarios $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$ (where K may be random) and choose as the best

$$\widehat{\mathbf{x}}^{\star} = \operatorname{argmin}_{i=1,2,\ldots,K} \widehat{\theta}(\mathbf{x}_i).$$

In other words, the algorithm returns the scenario with the sample best performance among all of those it simulated, and the estimated objective function value of the selected scenario is $\widehat{\theta}(\widehat{\mathbf{x}}^{\star})$. The number of feasible scenarios simulated will always be finite, since we have to stop eventually. In some cases K may exhaust C, so that we simulate all of the feasible scenarios, and in other cases $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$ is a subset of C that is generated by some sort of search.

As a specific example, consider a variation of the stochastic activity network (SAN) problem introduced in Sect. 3.4 and simulated in Sect. 4.4. The SO version of this problem is based on Henderson and Nelson (2006).

Recall that for the SAN

$$Y = \max\{A_1 + A_4, A_1 + A_3 + A_5, A_2 + A_5\}$$

is the total time to complete the project, where A_j is the time required for activity j (previously we denoted the activity times by X_j, but we use A_j here to avoid confusion with the scenario variable $\mathbf{x}$). Suppose that activity j is exponentially distributed with nominal mean τ_j, but we can reduce the mean to x_j at a cost of $c_j(\tau_j - x_j)$. In other words, c_j is the cost per unit reduction in the mean time of activity j. Thus, the project completion time as a function of the scenario $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5)$ is

$$Y(\mathbf{x}) = \max\{A_1(x_1) + A_4(x_4), A_1(x_1) + A_3(x_3) + A_5(x_5), A_2(x_2) + A_5(x_5)\},$$

where $A_j(x_j)$ is exponentially distributed with mean x_j. Let $\theta(x_1, x_2, \ldots, x_5) = \mathrm{E}(Y(\mathbf{x}))$, the expected value of the time to complete the project. If we have a fixed total budget b for the project, then the SO problem is

$$\begin{aligned} &\min\ \theta(x_1, x_2, \ldots, x_5) \\ &\sum_{j=1}^{5} c_j(\tau_j - x_j) \le b \\ &x_j \ge \ell_j,\ j = 1,2,3,4,5, \end{aligned} \tag{9.2}$$

where ℓ_j is the smallest achievable mean activity time for activity j. Notice that in this formulation we can actually save cost by letting $x_j > \tau_j$ for some activities.

The natural objective function estimator for scenario $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{i5})$ is

$$\widehat{\theta}(\mathbf{x}_i) = \frac{1}{n_i} \sum_{j=1}^{n_i} Y_j(\mathbf{x}_i),$$

where $Y_1(\mathbf{x}_i), Y_2(\mathbf{x}_i), \ldots$ are i.i.d. replications of the time to complete the project for scenario $\mathbf{x}_i$.

The size of the budget b, and the way in which we reduce the mean activity time to x_i, leads to very different problems:

- If we reduce the mean of activity i by allocating extra workers to the activity, and our budget is so tight that we can only afford to add at most one worker to at most one or two activities, then the number of feasible scenarios is small and we can likely simulate them all.
- If b is very large relative to the cost of an individual worker, and each worker assigned to activity i further reduces the mean time to complete the activity, then there might be so many feasible scenarios that we cannot simulate them all even though the number is finite. Therefore, the SO will require a search.
- If the reduction in mean activity time comes not from allocating workers, but from allocating some sort of capacity or power, then we might be able to treat x_i as continuous valued, implying that there might be uncountably many feasible scenarios.

We address all of these variations of SO in this chapter, as well as sensitivity analysis around a chosen solution and applying a change of measure to the outputs from a simulated solution to obtain estimates for an unsimulated one.

9.1 Errors in Simulation Optimization

No matter what algorithm we use to search for and simulate the collection of feasible scenarios $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$, there are three fundamental types of errors that can occur in SO problems:

1. *The optimal scenario is never simulated,* meaning
$$\mathbf{x}^\star \notin \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}.$$
This is also the case in deterministic nonlinear optimization problems when the feasible region cannot be exhausted, so we should not expect stochastic simulation optimization to be any easier.
2. *The best scenario that was simulated is not selected,* meaning
$$\widehat{\mathbf{x}}^\star \neq \mathbf{x}_B = \mathrm{argmin}_{i=1,2,\ldots,K}\, \theta(\mathbf{x}_i).$$
Because $\widehat{\theta}(\mathbf{x}_i)$ only estimates $\theta(\mathbf{x}_i)$, the best scenario that we simulated may not have the best *estimated* objective function value, and this causes us to select an inferior scenario. This error never occurs in deterministic optimization problems.
3. *The estimated objective function value of the selected scenario is not very accurate,* meaning
$$\left|\widehat{\theta}(\widehat{\mathbf{x}}^\star) - \theta(\widehat{\mathbf{x}}^\star)\right| \text{ is large.}$$

Because we select the scenario with the smallest *estimated* value, there is a natural bias toward scenarios whose estimated performance is better (smaller) than its true expected performance. Therefore, we tend to have an optimistic estimate of how good the selected scenario actually is; that is, we tend to have

$$\widehat{\theta}(\widehat{\mathbf{x}}^\star) < \theta(\widehat{\mathbf{x}}^\star).$$

Restating these errors in the context of the SAN optimization problem (9.2), they are:

1. The SO algorithm never discovers and simulates the best feasible setting of the mean activity times $\mathbf{x}^\star = (x_1^\star, x_2^\star, \ldots, x_5^\star)$.
2. The best setting actually simulated, $\mathbf{x}_B$, does not have the smallest estimated mean project completion time among $\widehat{\theta}(\mathbf{x}_i), i = 1, 2, \ldots, K$.
3. The sample mean activity time of the selected scenario $\widehat{\theta}(\widehat{\mathbf{x}}^\star)$ is not very close to its true mean project completion time $\theta(\widehat{\mathbf{x}}^\star)$.

Clearly we would like to control, measure, or eliminate Errors 1–3. Guaranteed asymptotic convergence (see Sect. 9.1.1) is the primary way SO algorithms are designed to address Error 1. Correct-selection guarantees address Error 2 and may also address Error 3; see Sect. 9.1.2.

9.1.1 Convergence

Convergence guarantees say something about what would happen if the SO algorithm was allowed to run forever. "Run forever" often implies exploring more and more feasible scenarios and simulating them more and more thoroughly. We focus on two particular definitions of convergence; for a broader discussion of convergence see Andradóttir (2006a).

The typical SO algorithm is iterative, and on the rth iteration it reports an estimated optimal scenario $\widehat{\mathbf{x}}_r^\star, r = 1, 2, \ldots$. An iteration could consist of simulating new feasible scenarios, or refining the estimates of scenarios that have already been simulated, or both. Notice that SO algorithms can and often do revisit scenarios that have already been simulated to better estimate their objective function values, a feature that makes no sense for deterministic optimization. As a result, it need not be the case that $K = r$ (the number of distinct scenarios simulated need not be the same as the number of iterations of the algorithm), and typically $K \ll r$.

A very natural definition of convergence for SO is

$$\Pr\left\{\lim_{r\to\infty} \widehat{\mathbf{x}}_r^\star = \mathbf{x}^\star\right\} = 1. \tag{9.3}$$

An algorithm with this property is *globally convergent* with probability 1. There exist globally convergent simulation optimization algorithms, particularly for the case when C contains a finite (even if very large) number of feasible scenarios. When the number of feasible scenarios is finite, convergence is usually proven by establishing that every feasible scenario will be simulated infinitely often in the limit. A good globally convergent algorithm assures global convergence even though it aggressively pursues improving scenarios. Although nothing like global convergence may actually happen in practice, it is reassuring to know that the algorithm would find the optimal scenario in the limit.

Next we consider *local convergence*. To do so we need the concept of a neighborhood of a feasible scenario $\mathbf{x}$, denoted by $\mathsf{N}(\mathbf{x})$. The neighborhood of $\mathbf{x}$ is a collection of feasible scenarios (so $\mathsf{N}(\mathbf{x}) \subset \mathsf{C}$) that are in some sense close to $\mathbf{x}$. For instance, if the components of $\mathbf{x}$ are integers (such as the initial level of inventory of d products), then one definition of a neighborhood is scenarios that differ from $\mathbf{x}$ by ± 1 in one component. That is, all scenarios of the form $\mathbf{x} \pm (0,0,\ldots,0,1,0,\ldots,0)$ that are also feasible. A scenario $\mathbf{x}'$ is *locally optimal* if

$$\theta(\mathbf{x}') \leq \theta(\mathbf{x}) \text{ for all } \mathbf{x} \in \mathsf{N}(\mathbf{x}');$$

that is, $\mathbf{x}'$ has as good or better (smaller) performance than the scenarios in its neighborhood. Of course, a scenario can be locally optimal but still not be very good overall.

Let $\mathsf{L} \subset \mathsf{C}$ be the set of locally optimal scenarios. Then local convergence is convergence to L. One definition is

$$\Pr\{\widehat{\mathbf{x}}_r^\star \notin \mathsf{L} \text{ infinitely often}\} = 0. \tag{9.4}$$

This means with probability 1 there is an iteration R such that for all iterations $r \geq R$ the estimated optimal scenario $\widehat{\mathbf{x}}_r^\star$ is a locally optimal scenario. Local convergence is less satisfying than global convergence, but it also does not require that all feasible scenarios are simulated even in the limit.

While a convergence guarantee, particularly a global convergence guarantee, is clearly a desirable property, a convergent algorithm will still, in the end, simulate only $K < \infty$ scenarios and return the one with the best estimated value. Correct-selection guarantees, described in the next section, relate to what can be said about the K simulated scenarios. When combined with a locally convergent algorithm, correct selection can provide a statistical guarantee of optimality.

9.1.2 Correct Selection

"Correct selection" primarily addresses Error 2. Let $\mathbf{x}_B = \operatorname{argmin}_{i=1,2,\ldots,K}\theta(\mathbf{x}_i)$ be the best scenario of those actually simulated by the SO algorithm; $\mathbf{x}_B = \mathbf{x}^\star$ only if the algorithm actually simulated $\mathbf{x}^\star$, and $\mathbf{x}_B$ may not even be locally optimal. However, once the SO algorithm stops then selecting $\mathbf{x}_B$ is the best that it can do.

Consider the correct-selection event

$$\begin{aligned}\mathsf{CS} &= \{\text{select } \mathbf{x}_B\} \\ &= \left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i), i = 1,2,\ldots,K, i \neq B\right\} \\ &= \bigcap_{i=1, i\neq B}^{K} \left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i)\right\}. \end{aligned} \tag{9.5}$$

CS requires that when the optimization algorithm terminates, the scenario with the best *estimated* performance is actually the best of the scenarios that were simulated. When this happens, we will make a correct selection, and we would like a guarantee that $\Pr\{\mathsf{CS}\}$ is high. Unfortunately, this sort of guarantee is affected by *multiplicity*, which is the effect that K has on the probability of an event like CS. To get some sense of this, consider two cases:

Independence: Suppose that the events $\left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i)\right\}$ are independent. Then

$$\Pr\{\mathsf{CS}\} = \prod_{i=1, i\neq B}^{K} \Pr\left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i)\right\}.$$

Thus, the more feasible scenarios $\mathbf{x}_i$ that we explore, the smaller the probability is that we select the correct one. For instance, if all of these events have probability $1-\alpha$, then $\Pr\{\mathsf{CS}\} = (1-\alpha)^{K-1}$. This is unfortunate because exploring broadly increases the chance of uncovering the optimal scenario $\mathbf{x}^\star$.

General but unknown dependence: Since the events $\left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i)\right\}$ all depend on the same $\widehat{\theta}(\mathbf{x}_B)$, it is unrealistic to think that they will be independent. When they are dependent, and we know nothing more than that, then the best we can say is that

$$\Pr\{\mathsf{CS}\} \geq 1 - \sum_{i=1, i\neq B}^{K} \Pr\left\{\widehat{\theta}(\mathbf{x}_B) \geq \widehat{\theta}(\mathbf{x}_i)\right\}.$$

This is even more discouraging than the independent case since this lower bound, which is known as the Bonferroni inequality, can become negative (and therefore meaningless) if K is large.[1]

The curse of multiplicity is that *exploring broadly (large K) makes it difficult to select correctly* because the statistical errors accumulate. There are two direct ways to mitigate this problem and both attack the probability of making a mistake, $\Pr\left\{\widehat{\theta}(\mathbf{x}_B) \geq \widehat{\theta}(\mathbf{x}_i)\right\}$.

[1] There are various expressions of the Bonferroni inequality, but the one we use here is $\Pr\{\cap_{i=1}^{k}\mathscr{E}_i\} \geq 1 - \sum_{i=1}^{k}\Pr\{\mathscr{E}_i^c\}$, where $\mathscr{E}_i$ is an event and $\mathscr{E}_i^c$ is its complement.

Suppose we can design our SO algorithm to guarantee that

$$\Pr\left\{\widehat{\theta}(\mathbf{x}_B) \geq \widehat{\theta}(\mathbf{x}_i)\right\} \leq \frac{\alpha}{K-1}$$

for all $\mathbf{x}_i \neq \mathbf{x}_B$. This means the chance of error in each comparison is very small. Then the Bonferroni inequality guarantees that $\Pr\{\mathsf{CS}\} \geq 1-(K-1)\alpha/(K-1) = 1-\alpha$. For instance, if our simulation optimization explores $K = 1{,}001$ feasible scenarios and we want 95% confidence, we have selected the best scenario of those simulated and then we can achieve this if

$$\Pr\left\{\widehat{\theta}(\mathbf{x}_B) \geq \widehat{\theta}(\mathbf{x}_i)\right\} \leq \frac{0.05}{1000} = 0.00005$$

for each alternative scenario $\mathbf{x}_i$. Not surprisingly, obtaining such a small level of error typically requires a very large expenditure of simulation effort (replications or run length), so this approach tends to be viable only when K is not too large.

The second approach tries to decrease $\Pr\left\{\widehat{\theta}(\mathbf{x}_B) \geq \widehat{\theta}(\mathbf{x}_i)\right\}$ by altering the joint distribution of $\widehat{\theta}(\mathbf{x}_i;\mathbf{U}_i), i = 1,2,\ldots,K$, where we now need to reintroduce the random numbers into the notation.

Suppose that we can represent the performance estimator as

$$\widehat{\theta}(\mathbf{x}_i;\mathbf{U}_i) = \theta(\mathbf{x}_i) + \varepsilon(\mathbf{U}_i),$$

where $\varepsilon(\mathbf{U}_i)$ is a mean-zero random variable that accounts for the fact that the simulation output is stochastic. Notice that in this stylized representation the simulation noise ε does not depend on the scenario $\mathbf{x}_i$. Therefore, if we use the same pseudorandom numbers for each scenario, $\mathbf{U}_1 = \cdots = \mathbf{U}_K = \mathbf{U}$, then

$$\begin{aligned}\Pr\left\{\widehat{\theta}(\mathbf{x}_B;\mathbf{U}) \geq \widehat{\theta}(\mathbf{x}_i;\mathbf{U})\right\} &= \Pr\{\theta(\mathbf{x}_B) + \varepsilon(\mathbf{U}) \geq \theta(\mathbf{x}_i) + \varepsilon(\mathbf{U})\} \\ &= \Pr\{\theta(\mathbf{x}_B) \geq \theta(\mathbf{x}_i)\} = 0.\end{aligned}$$

Thus, we never make a mistake no matter what K is!

We cannot expect the simulation noise to be independent of the scenario, or independent of the run length T_i or number of replications n_i for that matter. But this analysis does suggest that if the K scenarios behave similarly with respect to the random numbers, then we can decrease the probability of selection error by assigning the same random numbers to all scenarios. This technique is known as using *common random numbers* and we explore it more fully in Sect. 9.2.

How does the probability of correct selection relate to convergence? When a SO algorithm is globally convergent, but we cannot exhaust the feasible region C (as will always be the case when $\mathbf{x}$ is continuous valued), then a correct-selection guarantee only provides inference with respect to the scenarios $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$ that were actually simulated; it says nothing about the overall optimality. For a locally convergent algorithm, a correct-selection guarantee can provide confidence that $\widehat{\mathbf{x}}^\star$ is actually locally optimal provided all of its neighbors $\mathsf{N}(\widehat{\mathbf{x}}^\star)$ have been simulated.

Since the number of scenarios in $\mathsf{N}(\widehat{\mathbf{x}}^\star)$ is typically much smaller than the number of scenarios in C, simulating all neighbors is often possible. However, the inference is regarding local optimality, not global. Only when the number of feasible scenarios is small enough that we can simulate every $\mathbf{x} \in \mathsf{C}$ does correct selection relate directly to the overall optimality.

SO algorithms that do not exhaust the feasible scenarios C typically do not provide a correct-selection guarantee when they terminate. However, an (often small) amount of additional simulation of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$ can be used to add this guarantee and also control Error 3, which is that $\theta(\widehat{\mathbf{x}}^\star)$ is estimated badly. This strategy is known as "cleaning up" after the SO algorithm, and we describe one specific way to do it later. See also Boesel et al. (2003).

9.2 Random-Number Assignment

Having established the objectives in SO, we now turn to experiment design. Recall that we estimate the objective function $\theta(\mathbf{x})$ by $\widehat{\theta}(\mathbf{x}_i) = \widehat{\theta}(\mathbf{x}_i; T_i, n_i, \mathbf{U}_i)$. Experiment design consists of choosing the scenarios to simulate $\mathbf{x}_i, i = 1, 2, \ldots, K$, the allocation of simulation effort (n_i, T_i), and the assignment of random numbers $\mathbf{U}_i$ to scenario $\mathbf{x}_i$. We begin with random-number assignment.

The discussion of correct selection in Sect. 9.1.2 showed why the assignment of random numbers to the simulation of alternative scenarios should be a part of experiment design. Here we delve more deeply into that idea by first connecting correct selection to variance reduction and then describing how random-number assignment reduces variance.

We will assume two blocks of random numbers, say $\mathbf{U}_i$ and $\mathbf{U}_h$, are independent if $i \neq h$. As a practical matter we know that $\mathbf{U}_i$ is actually a block of deterministic pseudorandom numbers $\mathbf{u}_i = (u_{i1}, u_{i2}, \ldots, u_{is})$. We assume that s is so large that if we assign $\mathbf{u}_i$ to some purpose we will never use all s pseudorandom numbers in the block. We treat the random numbers $\mathbf{U}_i$ and $\mathbf{U}_h$ as independent because they actually correspond to nonoverlapping sequences of pseudorandom numbers $\mathbf{u}_i$ and $\mathbf{u}_h$. This allows us to "assign" either common or distinct blocks of random numbers to scenarios even though "assigning" anything that is truly random is a contradiction. The assignment is done by specifying seeds or streams as described in Sect. 6.5.3.

In Sect. 9.1.2 we focused on $\Pr\left\{\widehat{\theta}(\mathbf{x}_B) \geq \widehat{\theta}(\mathbf{x}_i)\right\}$, the probability that the estimated objective function value of the best scenario we simulated is larger than the estimated objective function value of a scenario that is actually inferior. Since we are minimizing, and since we select the scenario that appears to be the best, this event corresponds to an incorrect selection.

Suppose that these estimators are averages of a large number of more basic observations (e.g., replications) so that we are willing to treat them as normally distributed due to the central limit theorem. Then

$$\begin{aligned}
&\Pr\left\{\widehat{\theta}(\mathbf{x}_B) \ge \widehat{\theta}(\mathbf{x}_i)\right\} = \Pr\left\{\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i) \ge 0\right\} \\
&= \Pr\left\{\frac{\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i) - (\theta(\mathbf{x}_B) - \theta(\mathbf{x}_i))}{\sqrt{\mathrm{Var}\left[\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i)\right]}} \ge \frac{-(\theta(\mathbf{x}_B) - \theta(\mathbf{x}_i))}{\sqrt{\mathrm{Var}\left[\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i)\right]}}\right\} \\
&= \Pr\left\{Z \ge \frac{\theta(\mathbf{x}_i) - \theta(\mathbf{x}_B)}{\sqrt{\mathrm{Var}\left[\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i)\right]}}\right\}, \qquad (9.6)
\end{aligned}$$

where Z is a $\mathrm{N}(0,1)$ random variable. Since $\theta(\mathbf{x}_i) - \theta(\mathbf{x}_B) > 0$, Expression (9.6) is an increasing function of $\mathrm{Var}\left[\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i)\right]$. Stated differently, the smaller the $\mathrm{Var}\left[\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i)\right]$ is, the smaller the probability of an incorrect selection (or the greater the chance of a correct selection). This makes intuitive sense: the better our estimate of the difference $\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i)$ is, as measured by variance, the more likely we are to choose the one that is actually better.

The variance of the difference is[2]

$$\mathrm{Var}\left[\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i)\right] = \mathrm{Var}\left[\widehat{\theta}(\mathbf{x}_B)\right] + \mathrm{Var}\left[\widehat{\theta}(\mathbf{x}_i)\right] - 2\mathrm{Cov}\left[\widehat{\theta}(\mathbf{x}_B), \widehat{\theta}(\mathbf{x}_i)\right]. \qquad (9.7)$$

If each scenario $\mathbf{x}_i$ is assigned independent random numbers $\mathbf{U}_i$, then the estimators are independent and $\mathrm{Cov}\left[\widehat{\theta}(\mathbf{x}_i;\mathbf{U}_i), \widehat{\theta}(\mathbf{x}_B;\mathbf{U}_h)\right] = 0$. Common random numbers mean letting $\mathbf{U}_i = \mathbf{U}_B = \mathbf{U}$. The desired effect so as to increase the probability of correct selection is $\mathrm{Cov}\left[\widehat{\theta}(\mathbf{x}_i,\mathbf{U}), \widehat{\theta}(\mathbf{x}_B,\mathbf{U})\right] > 0$, which reduces the variance of the difference and increases the probability of correct selection. This is the connection between variance reduction and correct selection.

To gain some intuition, consider the output from the SAN simulation, rewritten in terms of the random numbers $(U_1, U_2, U_3, U_4, U_5)$ needed to generate the exponentially distributed activity times:

$$\begin{aligned}
Y(\mathbf{x}) &= \max\{A_1(x_1) + A_4(x_4), A_1(x_1) + A_3(x_3) + A_5(x_5), A_2(x_2) + A_5(x_5)\} \\
&= \max\{-\ln(1-U_1)x_1 - \ln(1-U_4)x_4, \\
&\qquad -\ln(1-U_1)x_1 - \ln(1-U_3)x_3 - \ln(1-U_5)x_5, \\
&\qquad -\ln(1-U_2)x_2 - \ln(1-U_5)x_5\},
\end{aligned}$$

[2] A basic result from mathematical statistics is that for random variables A and B, $\mathrm{Var}(A \pm B) = \mathrm{Var}(A) + \mathrm{Var}(B) \pm 2\mathrm{Cov}(A,B)$.

where x_j is the mean time for activity j. The use of common random numbers means that the simulations of scenarios $\mathbf{x}_i$ and $\mathbf{x}_h$ differ only in the values of x that multiply $-\ln(1-U)$. Thus, a collection of random numbers $(U_1, U_2, U_3, U_4, U_5)$ that make $Y(\mathbf{x}_i)$ larger than its expected value will tend to make $Y(\mathbf{x}_h)$ larger than its expected value also. This effect is particularly easy to see if for $\mathbf{x}_i$ we have $x_{i1} = x_{i2} = \cdots = x_{i5} = x_i$, and for $\mathbf{x}_h$ we have $x_{h1} = x_{h2} = \cdots = x_{h5} = x_h$. Then

$$\begin{aligned} Y(\mathbf{x}_h) &= \max\{-\ln(1-U_1)x_h - \ln(1-U_4)x_h, \\ &\qquad -\ln(1-U_1)x_h - \ln(1-U_3)x_h - \ln(1-U_5)x_h, \\ &\qquad -\ln(1-U_2)x_h - \ln(1-U_5)x_h\} \\ &= \frac{x_h}{x_i}\max\{-\ln(1-U_1)x_i - \ln(1-U_4)x_i, \\ &\qquad -\ln(1-U_1)x_i - \ln(1-U_3)x_i - \ln(1-U_5)x_i, \\ &\qquad -\ln(1-U_2)x_i - \ln(1-U_5)x_i\} \\ &= \frac{x_h}{x_i} Y(\mathbf{x}_i). \end{aligned}$$

Therefore,

$$\mathrm{Cov}[Y(\mathbf{x}_i), Y(\mathbf{x}_h)] = \frac{x_h}{x_i}\mathrm{Var}[Y(\mathbf{x}_i)] > 0.$$

Although it is more difficult to show mathematically, the covariance is still positive (although perhaps not as large) even when $\mathbf{x}_i$ and $\mathbf{x}_h$ do not have this special structure.

Two aspects of this example turn out to be important more generally: Notice that the output $Y(\mathbf{x})$ is monotonic in each of the random numbers $(U_1, U_2, U_3, U_4, U_5)$, and the same random number was used for the same purpose when simulating scenarios $\mathbf{x}_i$ and $\mathbf{x}_h$; e.g., we used U_j to generate activity time A_j in each scenario. Monotonicity is the result of two features of the simulation: (1) The inverse cdf method $A_j = F^{-1}(U_j)$ was used to generate the inputs A_j and it is always nondecreasing in U_j, and (2) the structure of the simulation itself caused the output to be monotonic in each of the inputs; that is, $Y = \max\{A_1 + A_4, A_1 + A_3 + A_5, A_2 + A_5\}$ is nondecreasing in each A_i. Monotonicity is central to inducing positive covariance, and synchronization (using the same random number for the same purpose in each scenario) tends to maximize the effect. This is shown formally in Glasserman and Yao (1992) but also makes intuitive sense because positive covariance means that the two quantities tend to go up and down together.

Sychronization is easy for the SAN simulation because each replication of each scenario uses exactly five random numbers. Thus, if the same starting seed or stream is assigned to each scenario, then the simulation of each scenario starts with the random number (say) U_1, and the jth replication uses the random numbers $U_{(j-1)5+1}, U_{(j-1)5+2}, U_{(j-1)5+3}, U_{(j-1)5+4}, U_{(j-1)5+5}$ to generate activity times A_1, A_2, A_3, A_4, A_5, respectively, regardless the value of $\mathbf{x}$.

Unfortunately, for other simulations synchronization is not as easy. To illustrate this, consider simulating two queueing systems that have the following features to compare their mean waiting times:

1. They have identical renewal arrival processes.
2. Scenario $x = 1$ has one fast server, while scenario $x = 2$ has two slower ones. Their service-time distributions are from the same family, but with different parameters.
3. Customers are not treated equally; instead, customers who are older are moved to the head of the queue. We have an input distribution for customer age.
4. We will simulate $n = 100$ replications of each scenario, and each replication ends after $T = 8$ h of simulated time.

Notice that the simulation does have some structural monotonicity: longer interarrival times lead to shorter waiting times (monotone decreasing); and longer service times lead to longer waiting times (monotone increasing). However, assigning the same starting seed or stream to each scenario is not sufficient here. Table 9.1 shows what could happen if we do so and just consume the random numbers (RNs) as needed by the simulation. Because the number of servers differs, rather quickly RNs are being used for different purposes. For instance, U_7 is used to generate a customer age in scenario 1 and an interarrival time in scenario 2. Therefore, the structural monotonicity of the queueing system is not fully exploited. Further, since the first replication of scenario 1 consumes fewer random numbers than scenario 2 (because fewer customers are able to be served in $T = 8$ h), the synchronization is even worse in the second replication. As we make more and more replications,

Table 9.1 A possible sequence of events and the random numbers consumed by the first and second replications of scenarios 1 and 2 using a single random-number stream

Scenario 1 (one server)			**Scenario 2 (two servers)**		
Event	RN	Purpose	Event	RN	Purpose
Arrival	U_1	Interarrival	Arrival	U_1	Interarrival
	U_2	Age		U_2	Age
	U_3	Service time		U_3	Service time
Arrival	U_4	Interarrival	Arrival	U_4	Interarrival
	U_5	Age		U_5	Age
				U_6	Service time
Arrival	U_6	Interarrival	Arrival	U_7	Interarrival
	U_7	Age		U_8	Age
End service	U_8	Service time	End service	U_9	Service time
⋮	⋮	⋮	⋮	⋮	⋮
End service	U_{123}	Service time	End service	U_{139}	Service time
End of replication 1			End of replication 1		
Arrival	U_{124}	Interarrival	Arrival	U_{140}	Interarrival
	U_{125}	Age		U_{141}	Age
	U_{126}	Service time		U_{142}	Service time
⋮	⋮	⋮	⋮	⋮	⋮

there will tend to be less and less synchronization, and smaller and smaller positive covariance.

The reason for pseudorandom-number generators having multiple starting seeds or streams is to facilitate synchronization. By assigning a distinct stream to each input process (arrivals, age, and service time), we guarantee that each pseudorandom number is used for the same purpose in each scenario. To maintain this synchronization across replications, we also need distinct starting streams or seeds for each replication. Thus, implementing common random numbers for the queueing example requires $100 \times 3 = 300$ starting seeds or streams for full synchronization, which is easily doable with a pseudorandom-number generator like `MRG32k3a` in L'Ecuyer (1999). See L'Ecuyer et al. (2002) for an efficient implementation of multiple streams.

Partial synchronization is better than no synchronization at all, but the benefits of full synchronization can be substantial. Shechter et al. (2006) describe a simulation that compares survival rates of an HIV cohort under two different treatment policies using independent simulations (different random numbers assigned to policies 1 and 2), and using common random numbers with partial synchronization and full synchronization. Relative to independent simulations, using common random numbers with full synchronization reduced the variance of the difference estimate by 93%, but only by 37% with partial synchronization. See Kelton (2006) for further discussion about synchronization strategies.

Clearly, variate generation via the inverse cdf also aids synchronization since each input variate is a function of one U. However, it is not always essential. In the queueing example the arrival-process input model is the same for scenarios 1 and 2. Thus, as long as the arrival process has a separate random-number stream, each scenario will experience exactly the same sequence of arrivals no matter what method is used to generate the interarrival times.

More care is required for the service times because the distribution changes. When the distribution changes but remains in the same family, it is sometimes possible to view the inputs for both scenarios as a transformation of a common base random variable. Examples include the normal distribution with mean μ and variance σ^2 since if Z is $N(0,1)$ then

$$\mu + \sigma Z$$

is $N(\mu, \sigma^2)$. Thus, as long as we generate the base Z using the same method and the same stream in both scenarios 1 and 2, we achieve full synchronization (and strong positive correlation). The same reasoning can be applied to lognormal variates, since they are transformations of normals.

Even when there is no simple expression for the inverse cdf, approximations and numerical inversion are possible (see Sect. 6.4). This will likely be slower than, say, a rejection method, but the additional effort may be worthwhile if a significant variance reduction can be achieved. When inversion is simply not possible, then Kachitvichyanukul and Schmeiser (1990) show that there are ways to create rejection-type algorithms that still maintain synchronization and monotonicity.

While our focus here has been on the effect of common random numbers on the probability of correct selection, the variance reduction is also relevant when the goal is simply to estimate the difference in performance between two or more scenarios. Suppose that we have just two scenarios, $\mathbf{x}_1$ and $\mathbf{x}_2$, for the mean activity times in the SAN simulation, and we simulate n replications of each: $Y_1(\mathbf{x}_1), Y_2(\mathbf{x}_1), \ldots, Y_n(\mathbf{x}_1)$ and $Y_1(\mathbf{x}_2), Y_2(\mathbf{x}_2), \ldots, Y_n(\mathbf{x}_2)$. Let $D_j = Y_j(\mathbf{x}_1) - Y_j(\mathbf{x}_2), j = 1, 2, \ldots, n$ be the differences between corresponding replications of the two scenarios. Then the paired-t confidence interval for the difference $\theta(\mathbf{x}_1) - \theta(\mathbf{x}_2)$ is

$$\overline{D} \pm t_{1-\alpha/2, n-1} \frac{S_D}{\sqrt{n}}, \tag{9.8}$$

where

$$\overline{D} = \frac{1}{n} \sum_{j=1}^{n} D_j$$

and

$$S_D^2 = \frac{1}{n-1} \sum_{j=1}^{n} (D_j - \overline{D})^2.$$

This CI is valid with or without common random numbers, provided that the D_j are approximately normally distributed or n is large. The effect of common random numbers shows up in the $\mathrm{Var}[D_j] = \mathrm{Var}[Y_j(\mathbf{x}_1) - Y_j(\mathbf{x}_2)]$, which is estimated by S_D^2. Thus, we expect S_D^2 to be smaller, making the CI shorter, when common random numbers are used, and this makes differences easier to detect.

The pairing of replications in (9.8) is important under common random numbers. If we have synchronized correctly, then $Y_j(\mathbf{x}_1)$ and $Y_j(\mathbf{x}_2)$ are dependent and positively correlated, while $Y_j(\mathbf{x}_1)$ and $Y_h(\mathbf{x}_2)$ are independent for different replications $j \neq h$. Therefore, we can treat the $D_1, D_2, \ldots, D_n$ as i.i.d.

For example, for the SAN we simulated two scenarios $\mathbf{x}_1 = (1, 1, 1, 1, 1)$ and $\mathbf{x}_2 = (0.5, 0.5, 1, 1.2, 1.2)$ with and without common random numbers. Based on 100 replications the estimated correlation between $Y_j(\mathbf{x}_1)$ and $Y_j(\mathbf{x}_2)$ was 0.93, reducing the half width of the confidence interval for the difference from ± 0.46 to ± 0.12 when the estimated difference is about $\overline{D} = 0.30$.

9.3 Ranking and Selection

Suppose that there are only K possible scenarios $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$, and we would like a guarantee of selecting the best among them. These scenarios arise either because there are exactly K feasible scenarios and all of them will be simulated, or because there are the K scenarios in C that were actually simulated during some sort of search. In the SAN optimization these would be K settings for the mean activity times that can be attained within the available budget.

SO among a finite number of scenarios is usually termed "ranking and selection," deriving its name from statistical procedures invented in the 1950s and 1960s for industrial and biostatistics applications; see Bechhofer et al. (1995). In the original context K was typically quite small. In the 1980s ranking and selection was embraced by simulation researchers, leading to rapid advancement in theory and widespread application; this section covers some basic foundations. Ranking-and-selection procedures are particularly amenable to parallelization, a topic discussed in a separate section of this chapter.

The formal proofs that the procedures presented in this section provide their stated guarantees assume that the output data from each scenario $\mathbf{x}_i$, $Y_1(\mathbf{x}_i), Y_2(\mathbf{x}_i), \ldots, Y_{n_i}(\mathbf{x}_i)$, are i.i.d. normally distributed with mean $\theta(\mathbf{x}_i)$ and finite variance. Independence is assured if they are the results from different replications and may be approximately correct if they are batch statistics from a single replication of a steady-state simulation (see Sect. 8.2). Further, in the procedures that follow $\theta(\mathbf{x}_i)$ is estimated by the sample mean

$$\widehat{\theta}(\mathbf{x}_i) = \overline{Y}(\mathbf{x}_i; n_i) = \frac{1}{n_i}\sum_{j=1}^{n_i} Y_j(\mathbf{x}_i).$$

As in Sect. 9.1.2, we let $\mathbf{x}_B$ denote the unknown best scenario; that is,

$$\theta(\mathbf{x}_B) = \min_{i=1,2,\ldots,K} \theta(\mathbf{x}_i).$$

There are numerous procedures available for many variations of this type of problem, and asymptotic justifications for cases in which normality and independence do not apply; some general references are Bechhofer et al. (1995), Frazier (2010), Goldsman and Nelson (1998) and Kim and Nelson (2006). We present two procedures that have been be useful in practice and that illustrate some key ideas.

9.3.1 Subset Selection

A subset selection procedure delivers a set of feasible scenarios $I \subseteq \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}$ with a guarantee that

$$\Pr\{\mathbf{x}_B \in I\} \geq 1 - \alpha.$$

In the best case I contains a single scenario, which is $\mathbf{x}_B$ with probability at least $1 - \alpha$. More typically, subset selection eliminates or screens out scenarios that (with high statistical confidence) are not the best so that attention can be focused on a much smaller group. The specific procedure below is particularly useful when

$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$ are the result of a search of C because it can allow the scenarios to have different numbers of observations.

1. Given $n_i \geq 2$ observations from scenario $\mathbf{x}_i$, set
$$t_i = t_{(1-\alpha)^{\frac{1}{K-1}},\, n_i-1}$$
the $(1-\alpha)^{\frac{1}{K-1}}$ quantile of the t distribution with $n_i - 1$ degrees of freedom, for $i = 1, 2, \ldots, K$.
2. Calculate the sample means $\overline{Y}(\mathbf{x}_i; n_i)$ and sample variances
$$S^2(\mathbf{x}_i) = \frac{1}{n_i - 1} \sum_{j=1}^{n_i} \left(Y_j(\mathbf{x}_i) - \overline{Y}(\mathbf{x}_i; n_i)\right)^2$$
for $i = 1, 2, \ldots, K$, and also the threshold
$$W_{ih} = \left(t_i^2 \frac{S^2(\mathbf{x}_i)}{n_i} + t_h^2 \frac{S^2(\mathbf{x}_h)}{n_h}\right)^{1/2}$$
for all $i \neq h$.
3. Form the subset
$$I = \left\{\mathbf{x}_i : \overline{Y}(\mathbf{x}_i; n_i) \leq \overline{Y}(\mathbf{x}_h; n_h) + W_{ih} \text{ for all } h \neq i\right\}.$$

The rule for which scenarios are included in the subset is simple: Include $\mathbf{x}_i$ if

$$\overline{Y}(\mathbf{x}_i; n_i) \leq \overline{Y}(\mathbf{x}_h; n_h) + W_{ih},\ h \neq i.$$

That is, we retain scenario $\mathbf{x}_i$ if its sample mean is smaller than all of the other sample means adjusted with a positive quantity W_{ih} that accounts for estimation error. Why does this work? The following argument is behind many subset selection procedures:

$$\begin{aligned}
&\Pr\{\mathbf{x}_B \in I\} \\
&= \Pr\left\{\overline{Y}(\mathbf{x}_B; n_B) \leq \overline{Y}(\mathbf{x}_h; n_h) + W_{Bh},\ h \neq B\right\} \\
&= \Pr\left\{\overline{Y}(\mathbf{x}_B; n_B) - \overline{Y}(\mathbf{x}_h; n_h) - [\theta(\mathbf{x}_B) - \theta(\mathbf{x}_h)] \leq W_{Bh} - [\theta(\mathbf{x}_B) - \theta(\mathbf{x}_h)],\ h \neq B\right\} \\
&\geq \Pr\left\{\overline{Y}(\mathbf{x}_B; n_B) - \overline{Y}(\mathbf{x}_h; n_h) - [\theta(\mathbf{x}_B) - \theta(\mathbf{x}_h)] \leq W_{Bh},\ h \neq B\right\}. \qquad (9.9)
\end{aligned}$$

The Inequality (9.9) follows because $-[\theta(\mathbf{x}_B) - \theta(\mathbf{x}_h)] \geq 0$, so removing it from the right-hand side makes the event more difficult to satisfy. The statistic

$$\overline{Y}(\mathbf{x}_i; n_i) - \overline{Y}(\mathbf{x}_h; n_h) - [\theta(\mathbf{x}_i) - \theta(\mathbf{x}_h)]$$

has mean 0 for all $i \neq h$, allowing the W_{ih}'s to be derived that give the desired probability based only on their variances.

The formal proof of validity is in Boesel et al. (2003), and it assumes that common random numbers are *not* used in the experiment design. This makes sense because the number of observations from each scenario can be different so the replications from different systems cannot be paired. If the number of observations is forced to be equal to n_0, say, and common random numbers are employed, then the procedure is valid if W_{ih} is replaced with

$$W'_{ih} = \left(t^2 \frac{S^2_{ih}}{n_0} \right)^{1/2}$$

with

$$t = t_{1-\alpha/(K-1), n_0 - 1}$$

and

$$S^2_{ih} = \frac{1}{n_0 - 1} \sum_{j=1}^{n_0} \left(Y_j(\mathbf{x}_i) - Y_j(\mathbf{x}_h) - [\overline{Y}(\mathbf{x}_i; n_0) - \overline{Y}(\mathbf{x}_h; n_0)] \right)^2 .$$

The effect of common random numbers is to reduce the size of the selected subset since it reduces the variance of the difference that in turn reduces W_{ih}. See Nelson et al. (2001).

9.3.2 Selection of the Best

Subset selection is an analysis method that takes whatever output data are available on the scenarios $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$ and provides inference on which ones are not the best. In this section we describe a design and analysis procedure that controls the amount of simulation effort expended and delivers a single selection at the end. To make this happen the user is required to specify the smallest difference in expected performance that is practically important, which we will call δ. Here are some examples:

In a queueing simulation where waiting times are on the order of tens of minutes, a difference of less than $1/2$ min (30 s) in mean waiting time might not be practically important, but a difference of more than that matters; therefore, we might set $\delta = 1/2$ min. In a reliability simulation where time to failure is on the order of weeks, differences between scenarios of 12 h or more in mean time to failure might be meaningful to the system designers; thus, we could set $\delta = 12$ h. And in a project planning optimization if the project takes a year or more to complete, then we might be satisfied with a scenario $\mathbf{x}_i$ that is not optimal if its mean time-to-completion $\theta(\mathbf{x}_i)$ is no more than $\delta = 7$ days longer than the optimal mean time $\theta(\mathbf{x}_B)$. *The choice of δ is entirely up to the user and depends on the situation.*

Each iteration of the procedure below takes a single observation from scenarios that have not yet been screened out, applies a type of subset selection, and continues until there is only one scenario in the subset. It is valid with or without common random numbers and can be very efficient (i.e., terminate quickly) if common random numbers are employed. The guarantee provided is that

$$\Pr\{\text{select } \mathbf{x}_B | \theta(\mathbf{x}_i) - \theta(\mathbf{x}_B) \geq \delta, \forall i \neq B\} \geq 1 - \alpha.$$

The specification of a practically significant difference δ is what allows the procedure to guarantee to terminate in finite time. Such procedures are called *indifference-zone selection procedures* and δ is what defines the indifference (not practically important) zone.

1. Specify a common first-stage number of observations $n_0 \geq 2$. Set

$$\eta = \frac{1}{2}\left[\left(\frac{2\alpha}{K-1}\right)^{-2/(n_0-1)} - 1\right].$$

2. Let $I = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}$ be the set of scenarios still in contention, and let $t^2 = 2\eta(n_0 - 1)$. Obtain n_0 observations $Y_j(\mathbf{x}_i), j = 1, 2, \ldots, n_0$ from each scenario $\mathbf{x}_i \in I$ and compute $\overline{Y}(\mathbf{x}_i; n_0)$. For all $i \neq h$ calculate

$$S_{ih}^2 = \frac{1}{n_0 - 1}\sum_{j=1}^{n_0}\left(Y_j(\mathbf{x}_i) - Y_j(\mathbf{x}_h) - \left[\overline{Y}(\mathbf{x}_i; n_0) - \overline{Y}(\mathbf{x}_h; n_0)\right]\right)^2,$$

the sample variance of the difference between scenarios i and h. Set $r = n_0$.
3. Set $I^{\text{old}} = I$. Let

$$I = \left\{\mathbf{x}_i : \mathbf{x}_i \in I^{\text{old}} \text{ and } \overline{Y}(\mathbf{x}_i; r) \leq \overline{Y}(\mathbf{x}_h; r) + W_{ih}(r), \forall h \in I^{\text{old}}, h \neq i\right\},$$

where

$$W_{ih}(r) = \max\left\{0, \frac{\delta}{2r}\left(\frac{t^2 S_{ih}^2}{\delta^2} - r\right)\right\}.$$

4. If $|I| = 1$, then stop and select the scenario in I as the best. Otherwise, take one additional observation $Y_{r+1}(\mathbf{x}_i)$ from each scenario $\mathbf{x}_i \in I$, update the sample means, set $r = r + 1$, and go to Step 3.

This procedure, due to Kim and Nelson (2001), applies subset selection iteratively as new observations are obtained, with the threshold $W_{ih}(r)$ shrinking to 0 so that eventually the scenario with the smallest sample mean is selected. Why does this work?

Recall the correct-selection event

$$\begin{aligned}
\mathsf{CS} &= \{\text{select } \mathbf{x}_B\} \\
&= \left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i), i = 1, 2, \ldots, K, i \neq B\right\} \\
&= \bigcap_{i=1, i \neq B}^{K}\left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i)\right\}
\end{aligned}$$

and consider the

$$\Pr\left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i)\right\}.$$

Since we operate under the assumption that $\theta(\mathbf{x}_i) - \theta(\mathbf{x}_B) \geq \delta$, we have

$$\begin{aligned}
&\Pr\left\{\widehat{\theta}(\mathbf{x}_B) < \widehat{\theta}(\mathbf{x}_i)\right\} \\
&= \Pr\left\{\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i) < 0\right\} \\
&= \Pr\left\{\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i) - [\theta(\mathbf{x}_B) - \theta(\mathbf{x}_i)] < -[\theta(\mathbf{x}_B) - \theta(\mathbf{x}_i)]\right\} \\
&\geq \Pr\left\{\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i) - [\theta(\mathbf{x}_B) - \theta(\mathbf{x}_i)] \leq \delta\right\}, \qquad (9.10)
\end{aligned}$$

where the Inequality (9.10) follows because we assume that the minimum difference is $\geq \delta$. As in subset selection, the statistic

$$\widehat{\theta}(\mathbf{x}_B) - \widehat{\theta}(\mathbf{x}_i) - (\theta(\mathbf{x}_B) - \theta(\mathbf{x}_i))$$

has mean 0, and since δ is known, the procedure can be designed to provide the desired guarantee by considering only δ, the variances, and the number of replications.

Remark 9.1. The indifference-zone formulation has dominated both the research and practice of ranking and selection. A natural question is what happens if there are one or more scenarios whose mean is within δ of the best? Clearly, the guarantee of selecting the single best is compromised, but one would hope it might carry over to selecting one of the δ-close alternatives; this is called a "good selection guarantee." Some procedures provide it, some do not, and for others the status is unknown. For instance, the procedure by Kim and Nelson (2001) above appears to select a good system in empirical evaluations, but there is no proof. See Eckman and Henderson (2018a) for what is known about good selection guarantees at this time. For a related but entirely indifference-zone-free formulation, see Fan et al. (2016).

9.3.3 Example

We illustrate these procedures with the SAN optimization, supposing that the project budget allows reducing the mean activity time for only one activity; the feasible scenarios are shown below:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$
x_{i1}	0.5	1	1	1	1
x_{i2}	1	0.5	1	1	1
x_{i3}	1	1	0.75	1	1
x_{i4}	1	1	1	0.5	1
x_{i5}	1	1	1	1	0.5

For subset selection, we independently simulated all $K = 5$ scenarios for $n = 100$ replications, giving the following summary statistics:

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$
$\overline{Y}(\mathbf{x}_i, n_i)$	3.045384364	3.556953882	3.313437416	3.361689108	2.772924336
$S^2(\mathbf{x}_i)$	2.318117804	2.627223846	2.946949109	2.915804886	2.099817647

Clearly, $\mathbf{x}_5$ has the smallest sample mean, but which others are close enough that they cannot be eliminated from being the best? For confidence level $1-\alpha = 0.95$, the subset selection procedure designed for use without common random numbers gives the selected subset as $I = \{\mathbf{x}_1, \mathbf{x}_5\}$. Why is, say, $\mathbf{x}_3$ eliminated but not $\mathbf{x}_1$?

Notice that $(1-\alpha)^{1/(K-1)} = 0.95^{1/4} = 0.987$, so the critical value for each scenario is $t_i^2 = t_{0.987,99}^2 = 5.145$. Scenario $\mathbf{x}_3$ is eliminated because

$$\begin{aligned}\overline{Y}(\mathbf{x}_3, n_3) &\approx 3.313 \not\leq \overline{Y}(\mathbf{x}_5, n_5) + \left(t_3^2 \frac{S^2(\mathbf{x}_3)}{n_3} + t_5^2 \frac{S^2(\mathbf{x}_5)}{n_5}\right)^{1/2} \\ &\approx 2.773 + \left(5.145 \frac{2.947}{100} + 5.145 \frac{2.100}{100}\right)^{1/2} \approx 3.282.\end{aligned}$$

However, $\mathbf{x}_1$ cannot be eliminated because

$$\overline{Y}(\mathbf{x}_1, n_1) \approx 3.045 \leq \overline{Y}(\mathbf{x}_5, n_5) + \left(t_1^2 \frac{S^2(\mathbf{x}_1)}{n_1} + t_5^2 \frac{S^2(\mathbf{x}_5)}{n_5}\right)^{1/2} \approx 3.250.$$

The guarantee is that the best scenario $\mathbf{x}_B$ is either $\mathbf{x}_1$ or $\mathbf{x}_5$ with 95% confidence. In fact, it is easy to see that $\theta(\mathbf{x}_1) = \theta(\mathbf{x}_5)$, and a bit more difficult to see that they are the best scenarios, so this is a correct selection.

Next we applied selection of the best at confidence level $1-\alpha = 0.95$, $n_0 = 10$ initial replications from each scenario, practically significant difference $\delta = 0.1$, and using common random numbers. The relevant constants are

$$\eta = \frac{1}{2}\left[\left(\frac{2\alpha}{K-1}\right)^{-2/(n_0-1)} - 1\right] = \frac{1}{2}\left[\left(\frac{2(0.05)}{4}\right)^{-2/9} - 1\right] \approx 0.635$$

and $t^2 = 2\eta(n_0 - 1) \approx 11.429$. At each iteration the procedure compares the sample means of all scenarios to each other. Comparison between, say, scenarios $\mathbf{x}_1$ and $\mathbf{x}_5$ requires the sample variance of the difference $S_{15}^2 \approx 0.294$, giving the decreasing threshold

$$\begin{aligned}W_{15}(r) &= \max\left\{0, \frac{\delta}{2r}\left(\frac{t^2 S_{15}^2}{\delta^2} - r\right)\right\} \\ &= \max\left\{0, \frac{0.1}{2r}(336.2 - r)\right\}.\end{aligned}$$

The thresholds $W_{12}(r), W_{13}(r)$, and $W_{14}(r)$ are computed in a similar fashion. We eliminate scenario $\mathbf{x}_1$ at iteration r if

$$\overline{Y}(\mathbf{x}_1;r) \not\leq \overline{Y}(\mathbf{x}_h;r) + W_{1h}(r)$$

for some other scenario $h \neq 1$.

The procedure selected $\mathbf{x}_5$ as the best after $171, 10, 33, 109, 171$ replications from $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5$, respectively. This means that $\mathbf{x}_2$ was eliminated after the initial sample, and then (in order) $\mathbf{x}_3, \mathbf{x}_4$ and $\mathbf{x}_1$ were eliminated. Why was scenario $\mathbf{x}_1$ eliminated if it is equivalent to $\mathbf{x}_5$? Because after 171 replications the procedure was able (with 95% confidence) to decide that $\mathbf{x}_5$ is either the best or within $\delta = 0.1$ of the best, and that is all that is required to make a correct selection.

9.3.4 The Rate-Optimal Allocation

From the procedures presented above it is clear that the efficiency of a ranking-and-selection procedure in the normally distributed output case depends (at least) on the means $\theta(\mathbf{x}_1), \theta(\mathbf{x}_2), \ldots, \theta(\mathbf{x}_K)$ and variances $\sigma^2(\mathbf{x}_1), \sigma^2(\mathbf{x}_2), \ldots, \sigma^2(\mathbf{x}_K)$ of the K scenarios, whether or not we employ common random numbers, and how we allocate the replications $n_1, n_2, \ldots, n_K$. By "efficiency" we mean the total number of replications required to reach a selection, or the size of the retained subset in subset selection. The means and variances are defined by the problem, as is (to some extent) the effect of common random numbers. What is under our control is the allocation of replications.

Previously we let $\overline{Y}(\mathbf{x}_i, n_i)$ denote the sample mean of system i when allocated n_i replications. Here we let N be the total number of replications to allocate, and write $n_i = \beta_i N$ where $\beta_i \geq 0$ and $\sum_{i=1}^{K} \beta_i = 1$. We ignore the obvious need for rounding the n_i to integers since we are going to let $N \to \infty$. What is the optimal choice for the β's?

To define "optimal" we need an objective. The one we consider here is having $\lim_{N\to\infty} \Pr\{\widehat{\mathbf{x}}^\star \neq \mathbf{x}_B\}$ go to 0 at the fastest possible rate, where $\widehat{\mathbf{x}}^\star$ is the selected scenario and $\mathbf{x}_B$ is the true best, which we assume is unique. Notice that $\Pr\{\widehat{\mathbf{x}}^\star \neq \mathbf{x}_B\}$ is the probability of an *incorrect* selection. Any allocation with all $\beta_i > 0$ will drive the probability of incorrect selection to 0 as $N \to \infty$; see Exercise 8. We are interested in the choice that sends it to 0 as rapidly as possible.

To ease the notation overload in what follows, we let i denote scenario $\mathbf{x}_i$, B denote the best scenario $\mathbf{x}_B$, and use subscripts like θ_i, σ_i, and $\overline{Y}_i$ rather than arguments like $\theta(\mathbf{x}_i)$, $\sigma(\mathbf{x}_i)$, and $\overline{Y}(\mathbf{x}_i, n_i)$.

Glynn and Juneja (2004) showed that in the normally distributed output case, when all scenarios are simulated independently, the rate-optimal β's satisfy the following balance equations:

$$\left(\frac{\beta_B}{\sigma_B}\right)^2 = \sum_{i \neq B} \left(\frac{\beta_i}{\sigma_i}\right)^2 \tag{9.11}$$

$$\frac{(\theta_i - \theta_B)^2}{\frac{\sigma_i^2}{\beta_i} + \frac{\sigma_B^2}{\beta_B}} = \frac{(\theta_j - \theta_B)^2}{\frac{\sigma_j^2}{\beta_j} + \frac{\sigma_B^2}{\beta_B}}, \quad \forall i, j \neq B. \tag{9.12}$$

Even though none of the terms in the equations above are known in practice, it is nevertheless a profound result as it characterizes how the true means and variances affect the optimal allocation, and it provides a target for implementable procedures to try to attain.

Exercise 9 asks you to show that in the special case that $\delta > 0$, $\theta_B = \theta - \delta$, $\theta_i = \theta$ for all $i \neq B$, and $\sigma_i^2 = \sigma^2$ for all i, the rate-optimal allocation is

$$\beta_i = \beta = \frac{1}{(K-1) + \sqrt{K-1}}, \ \forall i \neq B \tag{9.13}$$

$$\beta_B = \beta\sqrt{K-1}. \tag{9.14}$$

Notice that in this equal-variance indifference-zone-like case, the probability of incorrect selection is driven to 0 at the fastest rate if the best scenario is simulated $\sqrt{K-1}$ more often than the others.

Remark 9.2. Glynn and Juneja (2004) actually derive a more general result with (9.11)–(9.12) falling out for the special case of normally distributed outputs. The key is applying a large deviation result like Theorem 5.2. Here we use Theorem 5.2 to sketch out the approach for normally distributed output. This remark may be skipped without loss of continuity.

Consider the special case of $K = 2$ scenarios so that only (9.11) matters. Since we are minimizing, the probability of incorrect selection is $\Pr\{\overline{Y}_B - \overline{Y}_i \geq 0\}$ where $i \neq B$. Notice that

$$\overline{Y}_B - \overline{Y}_i \sim \mathrm{N}\left(\theta_B - \theta_i, \frac{1}{N}\left(\frac{\sigma_B^2}{\beta_B} + \frac{\sigma_i^2}{\beta_i}\right)\right)$$

and $\theta_B - \theta_i < 0$. From Cramér's Theorem 5.2

$$\lim_{N\to\infty} \frac{1}{N} \ln\left[\Pr\{\overline{Y}_B - \overline{Y}_i \geq 0\}\right] = -I(0) = -\frac{(\theta_B - \theta_i)^2}{2\left(\frac{\sigma_B^2}{\beta_B} + \frac{\sigma_i^2}{\beta_i}\right)},$$

where the last term on the right-hand side comes from applying $I(0)$ for the normal distribution. Therefore, the best possible rate results from maximizing $I(0)$, which is equivalent to minimizing the denominator $\sigma_B^2/\beta_B + \sigma_i^2/\beta_i$, where $\beta_B = 1 - \beta_i$. Exercise 10 asks you to show that the minimum occurs when $(\beta_B/\sigma_B)^2 = (\beta_i/\sigma_i)^2$, which matches (9.11).

9.3.5 Bayesian Procedures

There are two basic paradigms in designing for correct selection: frequentist (described above) and Bayesian. Bayesian ranking and selection consist of a sequence of decisions; the decisions include which scenario $\mathbf{x}$ to simulate next, and possibly whether or not to stop the procedure and select a scenario. The key to making good decisions is that in the Bayesian formulation uncertainty about $\theta(\mathbf{x})$ is represented as a prior probability distribution on its value, which is updated using Bayes rule to a posterior distribution as simulation observations are obtained. A good starting point for Bayesian ranking and selection is Frazier (2010).

An advantage of the Bayesian formulation is its flexibility; many kinds of information or knowledge about the problem can be incorporated into the prior beliefs, leading to substantial gains in efficiency. In addition, the procedure can be driven by different objectives: maximizing the posterior probability of correct selection and minimizing the expected opportunity cost (suboptimality) for the selected scenario under a given simulation budget are two examples. Bayesian procedures are particularly appropriate for fixed-budget settings.

No procedure, frequentist or Bayesian, can dominate in all situations. Bayesian procedures are often more efficient in terms of the total number of observations required to make a selection but the algorithm itself may have substantial computational overhead and may not be easily implemented in parallel. Bayesian procedures do not provide (or intend to provide) the correct-selection guarantee that the frequentist procedures do. However, recent advances have shown strong links between Bayesian inspired procedures and desirable frequentist behavior; see Chen and Ryzhov (2019), Frazier (2014), and the discussion that follows.

Bayesian statistics is a deep, subtle, and important topic not easily summarized. The goal here is to provide a gentle introduction that is easy to follow by leaving out lots of details and also to present a procedure that gives the flavor of a Bayesian approach and its connection to frequentist probability of correct selection.

The Bayesian formulation begins by treating a specific problem with unknown means $\theta(\mathbf{x}_1), \theta(\mathbf{x}_2), \ldots, \theta(\mathbf{x}_K)$ as an instance of a random problem $\Theta(\mathbf{x}_1), \Theta(\mathbf{x}_2), \ldots, \Theta(\mathbf{x}_K)$, where the capital Θ indicates a random variable. The joint probability distribution of $\Theta(\mathbf{x}_1), \Theta(\mathbf{x}_2), \ldots, \Theta(\mathbf{x}_K)$ provided by the analyst is called the *prior*. In what follows we assume that the variances $\sigma^2(\mathbf{x}_1), \sigma^2(\mathbf{x}_2), \ldots, \sigma^2(\mathbf{x}_K)$ are known, for simplicity.

Of course the specific problem is not actually random—the true means are implied by the simulation code—but uncertainty about the problem is captured in the prior. After obtaining simulation outputs from one or more scenarios, Bayes rule provides the mathematical and computational machinery to reduce our uncertainty about the problem by updating the prior given the new information; the update is called the *posterior*. The posterior then provides guidance as to what scenarios to simulate next. Obtaining additional outputs and updating the posterior continue until either we run out of time or the posterior implies sufficient confidence to select a scenario.

To describe an algorithm, let $\mathscr{H}$ denote the history of the algorithm up to the current iteration, by which we mean a record of which $\mathbf{x}$'s were simulated and the outputs $Y(\mathbf{x})$ that were obtained. By "posterior" we mean the conditional distribution of $\{\Theta(\mathbf{x}_1),\Theta(\mathbf{x}_2),\ldots,\Theta(\mathbf{x}_K)\}$ given the prior and $\mathscr{H}$. Since the posterior is a distribution, we can compute probabilities, expectations, etc. with respect to it. One posterior summary measure that has seen wide use is *expected improvement* (EI). Let $\mathbf{x}_i$ and $\mathbf{x}_j$ be two scenarios. For a minimization problem the EI of selecting $\mathbf{x}_i$ over selecting $\mathbf{x}_j$ is

$$\mathrm{E}\left[\max\{0,\Theta(\mathbf{x}_j)-\Theta(\mathbf{x}_i)\}|\mathscr{H}\right]. \tag{9.15}$$

Let us examine (9.15) carefully: $\Theta(\mathbf{x}_i)$ and $\Theta(\mathbf{x}_j)$ are random variables representing our uncertainty about the performance of scenarios $\mathbf{x}_i$ and $\mathbf{x}_j$. In a minimization problem, $\max\{0,\Theta(\mathbf{x}_j)-\Theta(\mathbf{x}_i)\}$ is how much smaller $\Theta(\mathbf{x}_i)$ is than $\Theta(\mathbf{x}_j)$, if it is smaller. EI averages this over the conditional (posterior) distribution of $\Theta(\mathbf{x}_i)$ and $\Theta(\mathbf{x}_j)$, given the history. A scenario with large EI relative to other solutions is a solution worth simulating to obtain a better estimate of its performance. EI was originally introduced in Jones et al. (1998) for deterministic computer experiments, and extended to stochastic simulation in the form we use it here by Salemi et al. (2019).

The following ranking-and-selection algorithm due to Chen and Ryzhov (2019) is called mCEI; it is easily modified to allow unknown variances. In mCEI one scenario is simulated on each iteration: either the solution with the largest EI relative to the sample best solution, or the sample best solution if its number of replications is out of balance. In the algorithm $n(\mathbf{x})$ is the number of replications obtained from solution $\mathbf{x}$ up through the current iteration.

1. Simulate all scenarios for $n_0 \ge 1$ replications, let $\overline{Y}(\mathbf{x}_1),\overline{Y}(\mathbf{x}_2),\ldots,\overline{Y}(\mathbf{x}_K)$ be the sample means, and let $\widehat{\mathbf{x}}^\star = \mathrm{argmin}_{\mathbf{x}}\overline{Y}(\mathbf{x})$, the scenario with the best sample mean. Update $\mathscr{H}$.
2. Let $\mathbf{x}' = \mathrm{argmax}_{\mathbf{x}\neq\widehat{\mathbf{x}}^\star}\mathrm{E}\left[\max\{0,\Theta(\widehat{\mathbf{x}}^\star)-\Theta(\mathbf{x})\}|\mathscr{H}\right]$, the scenario with the largest EI relative to $\widehat{\mathbf{x}}^\star$.
3. Check whether
$$\left(\frac{n(\widehat{\mathbf{x}}^\star)}{\sigma(\widehat{\mathbf{x}}^\star)}\right)^2 < \sum_{\mathbf{x}\neq\widehat{\mathbf{x}}^\star}\left(\frac{n(\mathbf{x})}{\sigma(\mathbf{x})}\right)^2.$$
If yes, then simulate $\widehat{\mathbf{x}}^\star$; otherwise simulate $\mathbf{x}'$.
4. Update the sample means and $\mathscr{H}$.
5. Let $\widehat{\mathbf{x}}^\star = \mathrm{argmin}_{\mathbf{x}}\overline{Y}(\mathbf{x})$. If the budget has been expended, then return $\widehat{\mathbf{x}}^\star$ as the selected solution; otherwise go to Step 2.

Remarkably, Chen and Ryzhov (2019) show that mCEI, which is inspired by Bayesian reasoning, converges to the frequentist rate-optimal allocation of Glynn and Juneja (2004) as the budget goes to infinity. This not only indicates that it is efficient with respect to frequentist probabililty of incorrect selection, but also provides a connection between frequentist and Bayesian perspectives.

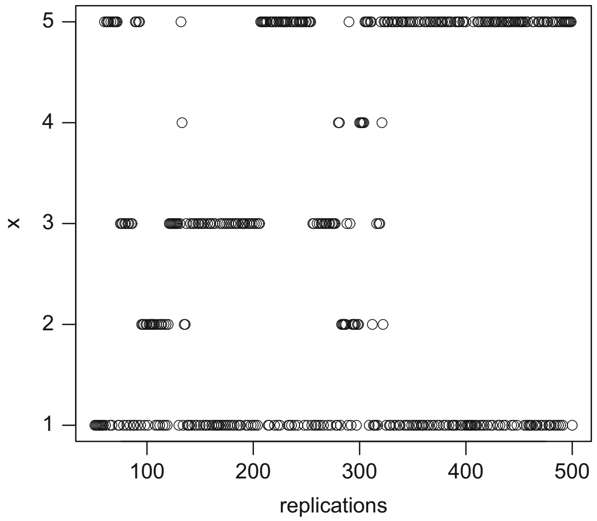

Fig. 9.1 A sample path of the mCEI algorithm applied to the SAN problem. Each ∘ represents one replication obtained from scenario $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_5$. Initially $n_0 = 10$ replications are obtained from each scenario

Recall the SAN example in Sect. 9.3.3 with $K = 5$ scenarios. Figure 9.1 shows one sample path of a modified mCEI algorithm in which $n_0 = 10$ initial replications are used to estimate the means and variances, and both of these are updated as additional replications are obtained. The budget is 500 replications, the same as given to the subset procedure, and nearly the same as consumed by the select-the-best procedure, in Sect. 9.3.3. Recall that $\mathbf{x}_1$ and $\mathbf{x}_5$ are statistically equivalent and superior to the others. Notice that the two superior systems are simulated more frequently, but no systems are completely eliminated.

Remark 9.3. EI is just one of many possible ways to exploit the posterior distribution to drive a ranking-and-selection procedure. Functionals like EI are sometimes called "acquisition functions" because they specify which scenarios to simulate (acquire) next. Critical is that the acquisition function can actually be computed with respect to the posterior. Exercise 11 asks you to show that calculating EI for the case of Chen and Ryzhov (2019) is easy.

9.3.6 Cleaning Up After Simulation Optimization

The procedures described in this section can be used for simulation optimization when $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}$ are the only feasible scenarios and all of them will be simulated. When this is the case, the three types of the SO error—failure to simulate the optimal scenario, failure to select-the-best scenario that was simulated, and poorly estimating the performance of the selected scenario—can all be managed.

On the other hand, when $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K\}$ is only a subset of the feasible scenarios, then failure to simulate the optimal scenario can only be addressed in an asymptotic sense, as described in the discussion on convergence in Sect. 9.1.1. The other two SO errors can, however, still be controlled. We will focus on the second one, selection error, since it is the most critical.

At the end of a SO search, the output data that have been generated are

$$Y_j(\mathbf{x}_i),\ j = 1, 2, \ldots, n_i,\ i = 1, 2, \ldots, K. \tag{9.16}$$

Since SO algorithms often revisit scenarios and accumulate observations on each visit, there is no reason to believe that the n_i's will be equal, which limits the effectiveness of common random numbers.

Boesel et al. (2003) proposed the idea of "cleaning up" after an SO algorithm terminates by starting with the data (9.16) and doing as little additional simulation as possible to obtain a correct-selection guarantee. The approach consists of two steps:

1. Apply a subset selection procedure to (9.16) to reduce (typically greatly) the number of scenarios that are competitive to be the best. The subset selection procedure described Sect. 9.3.1 is particularly useful for clean up because it can start from unequal sample sizes n_i, and it only requires that the sample means and sample variances from all K scenarios be retained, not all of the raw data.
2. Apply a selection-of-the-best procedure to the scenarios in the subset. The procedure presented here could be used, but others described in Boesel et al. (2003) are easier to apply.

Boesel et al. (2003) showed that certain combinations of subset selection and selection of the best can be formally proven to deliver an overall correct-selection guarantee. Xu et al. (2010) later showed that a precise estimation guarantee (the third SO error) can be added in some cases. However, the analysis in these papers assumes that the data in (9.16) satisfy the first-stage-sample assumptions of the subset or selection-of-the-best procedure. Eckman and Henderson (2018b) note that when these outputs are obtained by running an optimization algorithm, then the assumptions may not be strictly satisfied, which can compromise the statistical guarantees. Nevertheless, even applied informally clean-up will greatly improve SO performance.

Remark 9.4. Instead of "cleaning up" one might hope to manage selection error at every iteration of the SO search, so that whenever the algorithm terminates there is a statistical guarantee that the current sample best scenario is the best of those simulated up to that iteration. Hong and Nelson (2007a) showed that this is possible but computationally expensive unless K is small. Rapid progress in uncovering better and better scenarios in a large feasible space C typically requires less simulation effort than is required to guarantee that the sample best scenario is the best at all iterations. This is why we recommend the clean-up approach *after* the SO search.

9.4 Adaptive Random Search

We now consider simulation optimization problems such as (9.1) for which C is discrete and finite but contains far too many scenarios to simulate them all. Adaptive random search provides a framework for attacking such problems within which asymptotic convergence can be proven while still allowing for algorithms that aggressively pursue improving scenarios. We first describe adaptive random search at a high level and then provide a specific algorithm.

An adaptive random search algorithm is iterative, with the iterations indexed by $i = 1, 2, \ldots$. On the ith iteration, the algorithm has a probability distribution $P_i(\mathbf{x})$ on the scenarios $\mathbf{x} \in \mathsf{C}$. This distribution is used to sample one or more scenarios from C. The distribution $P_i(\cdot)$ may (and usually will) depend on its "memory" of some or all of the scenarios that have been simulated in previous iterations and perhaps the output data obtained from simulating them.

The distinction between *sampling* scenarios and *simulating* them is important: "Sampling" means randomly choosing a scenario $\mathbf{x}$ from C, while "simulating" a scenario means generating output performance data. Adaptive random search algorithms have a simulation allocation rule that specifies how much simulation effort (e.g., how many replications) to expend on a sampled scenario, and also a value $V(\mathbf{x})$ for each simulated scenario. Most often $V(x) = \widehat{\theta}(\mathbf{x})$, its estimated performance, but it can be other measures such as the number of times $\mathbf{x}$ has been visited by the search.

Initialize: Set the iteration counter to $i = 1$.

Sample: Choose an estimation set, which is a collection of scenarios $\mathsf{E}_i \subset \mathsf{C}$ where some or all of the scenarios were chosen by sampling according to $P_i(\cdot)$ and others were retained from previous iterations.

Simulate: Apply the simulation allocation rule to simulate the scenarios $\mathbf{x} \in \mathsf{E}_i$.

Evaluate: Update the value $V(\mathbf{x})$ for all $\mathbf{x} \in \mathsf{E}_i$ and choose as $\widehat{\mathbf{x}}^\star_{i+1}$ the scenario with the best value $V(\mathbf{x})$.

Iterate: Update the algorithm memory, let $i = i + 1$, and go to **Sample**.

Notice that the algorithm contains no stopping rule, which makes sense for proving asymptotic convergence. When a specific algorithm is applied, stopping may occur when the simulation budget is exhausted or when progress appears to slow.

The sampling distribution $P_i(\cdot)$ and simulation allocation rule work together to guarantee asymptotic convergence. When no strong structural properties of $\theta(\mathbf{x})$ are known, then global convergence usually requires that all scenarios in C are simulated infinitely often in the limit. For local convergence a scenario and all of its neighbors must be simulated infinitely often for any scenario that appears to be locally optimal. See Andradóttir (1999, 2006b) and Hong and Nelson (2007a) for specific conditions that insure global and local convergence, respectively, of adaptive random search algorithms.

Three types of sampling distributions $P_i(\cdot)$ are common:

- A distribution that puts positive probability on a small number of feasible scenarios in a neighborhood of $\widehat{\mathbf{x}}_i^\star$. Typically, $P_i(\cdot)$ and the neighborhood structure connect C so that any scenario is reachable from any other scenario after a sufficient number of iterations.
- A distribution that puts positive probability on a "promising" subset of C that may be large or small but is not necessarily a neighborhood of $\widehat{\mathbf{x}}_i^\star$. Typically such distributions use memory in an intelligent way to concentrate the search. The algorithm described below is of this type.
- A distribution that puts positive probability on all of C. Typically the distribution changes as a function of the iteration and the memory, focusing probabilistically on promising regions of C.

To illustrate some of the key ideas in adaptive random search we describe the adaptive hyperbox algorithm (AHA) of Xu et al. (2012). This algorithm is applicable when the components of $\mathbf{x}$ are integer-ordered decision variables and the neighborhood of $\mathbf{x}$ is defined to be all feasible scenarios that differ from $\mathbf{x}$ by ± 1 in one component. For instance, in two dimensions the scenario $\mathbf{x} = (3,7)$ has neighbors $\{(2,7),(4,7),(3,6),(3,8)\}$. Under mild conditions AHA can be shown to converge to a locally optimal scenario (i.e., a scenario that is better than all of its feasible neighbors).

Here is the AHA implementation of adaptive random search:

Value: AHA uses the estimated objective function value $\widehat{\theta}(\mathbf{x})$ accumulated over all output data generated for scenario $\mathbf{x}$. The key technical assumption is that $\widehat{\theta}(\mathbf{x})$ converges with probability 1 to $\theta(\mathbf{x})$ as the amount of simulation effort (replications or run length) goes to infinity.

Memory: AHA remembers all scenarios that have been simulated during the course of the search and their estimated objective function values accumulated over all output data generated.

Distribution $P_i(\cdot)$: AHA puts positive probability on scenarios that are feasible and are in or on a hyperbox that surrounds $\widehat{\mathbf{x}}_i^\star$, the simulated scenario with the best estimated objective function value on iteration i. The hyperbox is defined by scenarios that have been simulated and are closest to $\widehat{\mathbf{x}}_i^\star$ in one or more coordinate. All other feasible scenarios have probability 0. A fixed number of scenarios are sampled on each iteration, and revisiting scenarios is allowed.

Estimation set E_i: For the estimation set AHA includes the current sample best $\widehat{\mathbf{x}}_i^\star$ and any scenarios sampled from $P_i(\cdot)$.

Simulation allocation rule: AHA requires that whenever a scenario is in the estimation set it must receive additional simulation (replications or run length).

Figure 9.2 illustrates how AHA works in a feasible region similar to the SAN optimization problem except that it is only in two dimensions. On the first iteration the three scenarios indicated by $\bullet$ are randomly sampled from within the feasible C, all are simulated according to the simulation allocation rule, and the scenario indicated by $\widehat{\mathbf{x}}_1^\star$ is the sample best. Therefore, the other two $\bullet$ scenarios are used to build the hyperbox. Scenarios for the next iteration are sampled from the intersection

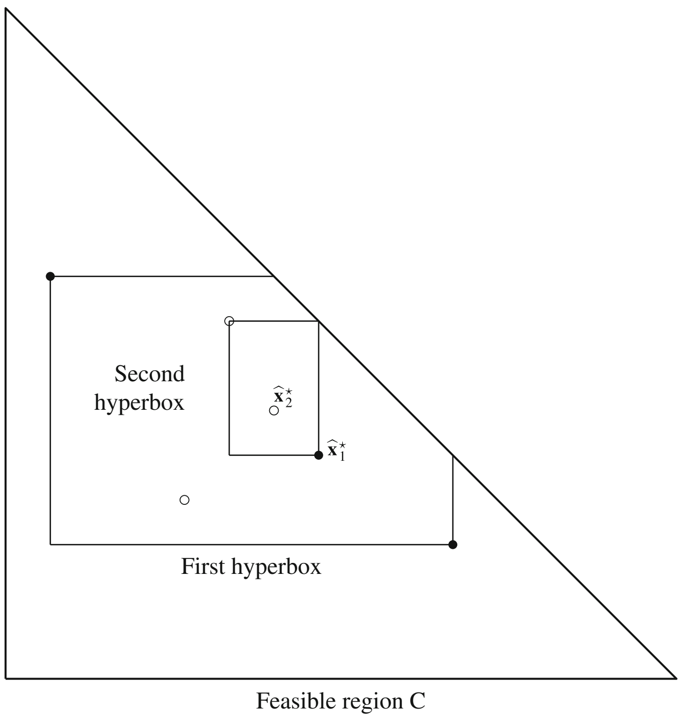
Second hyperbox
$\widehat{\mathbf{x}}_2^\star$
$\widehat{\mathbf{x}}_1^\star$
First hyperbox
Feasible region C

Fig. 9.2 First iteration (● scenarios) and second iteration (○ scenarios) of the adaptive hyperbox algorithm

of the hyperbox with C; this provides the ○ scenarios that are simulated along with $\widehat{\mathbf{x}}_1^\star$ according to the simulation allocation rule. Of these four scenarios, the one indicated by $\widehat{\mathbf{x}}_2^\star$ is now the sample best, so the scenarios that are closest in each coordinate direction are used to define the second hyperbox.

It is easy to see that if there was no simulation error ($\widehat{\theta}(\mathbf{x}) = \theta(\mathbf{x})$), then the hyperbox would continue to shrink until it surrounds a single scenario with $\theta(\mathbf{x}^\star)$ value less than or equal to all of its neighbors; that is, a locally optimal scenario. Even with simulation error this will happen in the limit because the estimates $\widehat{\theta}(\mathbf{x})$ of an apparently locally optimal scenario and its neighbors converge to their true expected values as the number of iterations (and therefore simulation effort) goes to infinity.

9.5 Statistical Learning

We now consider simulation optimization problems such as (9.1) for which the feasible region C is (typically) a bounded convex subset of $\Re^d$. Examples include a C that contains the integer values in a bounded convex region, or one that is continuous and defined by (say) linear constraints. In any event C contains far too many scenarios to simulate all of them.

Section 9.4 described adaptive random search, which leads to algorithms that can be proven to converge to a globally or locally optimal scenario in the limit, but may have disappointing finite time performance, especially when the budget is tight relative to the size of C. The rationale for adaptive random search is the belief that good scenarios tend to cluster in C, so whenever a good scenario is discovered one should concentrate the search around it. What we call "statistical learning" makes formal, and stronger, assumptions about the spatial relationship among feasible scenarios and exploits it in hopes of making more rapid progress. The formalism we exploit here is the spatial Gaussian process (GP), as described in Sect. 5.2.5.3; however, others are possible including polynomial regression models; see, for instance, Chapter 4 in Fu (2015).

At a high level, the idea is simple: Treat the unknown objective function $\theta(\mathbf{x})$ as a realization of a GP $G(\mathbf{x})$ with mean function $\mathsf{m}(\mathbf{x}) = \mathrm{E}[G(\mathbf{x})]$ and covariance function $\mathsf{C}(\mathbf{x},\mathbf{x}') = \mathrm{Cov}[G(\mathbf{x}), G(\mathbf{x}')]$. As we search C by simulating scenarios $\mathbf{x}$ and observing noisy estimates of their performance, $Y(\mathbf{x}) = G(\mathbf{x}) + \varepsilon(\mathbf{x})$, we update the conditional distribution of $G(\cdot)$ using Eqs. (5.14)–(5.15) and refine the search based on this learned information. For instance, choosing the next scenario to simulate via expected improvement (EI) as described in Sect. 9.3.5 still applies. This generic approach is sometimes called Bayesian optimization.

Imposing spatial covariance among all scenarios allows the search to learn about unsimulated scenarios via the ones that have been simulated and therefore is ideal for search. Although choosing the prior mean function to be an unknown constant, $\mathsf{m}(\mathbf{x}) = \mathsf{m}$, seems to work well, the choice of covariance function $\mathsf{C}(\cdot,\cdot)$ is critical as it implies properties of the objective function surface $\theta(\mathbf{x})$. For instance, the popular squared-exponential covariance function

$$\mathsf{C}(\mathbf{x},\mathbf{x}') = \tau^2 \exp\left(\sum_{i=1}^{d} \beta_i (x_i - x_i')^2\right) \tag{9.17}$$

implies that $\theta(\mathbf{x})$ is infinitely continuously differentiable in $\Re^d$; see Santner et al. (2003). Notice also that (9.17) has $d+1$ tuning parameters $\tau, \beta_1, \beta_2, \ldots, \beta_d$ whose values (along with m) are typically estimated from an initial experiment; their values also matter.

Simulation optimization via statistical learning is an active research area with new developments regularly. Excellent tutorials include Frazier (2010, 2018).

9.6 Searching in Improving Directions

We now consider simulation optimization problems such as (9.1) for which the objective function $\theta(\mathbf{x})$ is continuous and differentiable in $\mathbf{x}$, and C is (typically) a convex subset of $\Re^d$. Many algorithms for deterministic nonlinear optimization problems search C for (at least locally) optimal scenarios by moving in improving directions based on the gradient of $\theta(\mathbf{x})$:

$$\nabla\theta(\mathbf{x}) = \left(\frac{\partial\theta(\mathbf{x})}{\partial x_1}, \frac{\partial\theta(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial\theta(\mathbf{x})}{\partial x_d}\right).$$

For simulation optimization we can try to estimate $\nabla\theta(\mathbf{x})$, just as we estimate $\theta(\mathbf{x})$ itself, and then incorporate it into some sort of steepest descent algorithm such as stochastic approximation (e.g., Fu (2006)). Gradient estimators are also useful in their own right for sensitivity analysis; see Sect. 9.10. Our approach in this section is to provide enough understanding of gradient estimation so that the reader can navigate the vast literature on this subject. More so than any other topic in this book, the mathematical conditions required for provable correctness of gradient estimators are difficult to explain and verify; good general references are Fu (2006) and L'Ecuyer (1990).

Recall that the basic output from the SAN optimization (9.2) can be expressed in two ways:

$$\begin{aligned} Y(\mathbf{x}) &= \max\{A_1(x_1)+A_4(x_4), \\ &\qquad A_1(x_1)+A_3(x_3)+A_5(x_5), \\ &\qquad A_2(x_2)+A_5(x_5)\} &\quad (9.18) \\ &= \max\{-\ln(1-U_1)x_1-\ln(1-U_4)x_4, \\ &\qquad -\ln(1-U_1)x_1-\ln(1-U_3)x_3-\ln(1-U_5)x_5, \\ &\qquad -\ln(1-U_2)x_2-\ln(1-U_5)x_5\}, &\quad (9.19) \end{aligned}$$

where (U_1,U_2,U_3,U_4,U_5) are the pseudorandom numbers needed to generate the exponentially distributed activity times $(A_1,A_2,\ldots,A_5)$, and $\theta(\mathbf{x}) = \mathrm{E}(Y(\mathbf{x}))$. We will illustrate key ideas for gradient estimation by estimating the first component of the gradient, $\partial\theta(\mathbf{x})/\partial x_1$, which is the partial derivative of the mean time to complete the project with respect to the mean of the first activity time, x_1. Precisely the same ideas apply to the other components x_2,x_3,x_4,x_5.

9.6.1 Finite Differences

Representation (9.19) and the definition of derivative suggest a natural approximation. Let

$$\begin{aligned} Y(\mathbf{x}+\Delta x_1) = \max\{&-\ln(1-U_1)(x_1+\Delta x_1)-\ln(1-U_4)x_4, \\ &-\ln(1-U_1)(x_1+\Delta x_1)-\ln(1-U_3)x_3-\ln(1-U_5)x_5, \\ &-\ln(1-U_2)x_2-\ln(1-U_5)x_5\}, \end{aligned}$$

which is the time to complete the project when the mean time for activity 1 is $x_1 + \Delta x_1$ (the term Δx_1 is used to denote the vector $(\Delta x_1, 0, 0, \ldots, 0)^\top$). Then form the finite-difference (FD) estimator

$$\mathrm{FD}(x_1) = \frac{Y(\mathbf{x}+\Delta x_1) - Y(\mathbf{x})}{\Delta x_1}. \tag{9.20}$$

Notice that we intentionally used common random numbers—the same $(U_1, U_2, U_3, U_4, U_5)$ for each scenario—which makes sense because $\mathrm{FD}(x_1)$ estimates the difference between the mean time to complete the project for scenarios $\mathbf{x} + \Delta x_1$ and $\mathbf{x}$. Typically we would average $\mathrm{FD}(x_1)$ estimators from across $n > 1$ replications to estimate $\partial\theta(\mathbf{x})/\partial x_1$.

FD gradient estimators are easy to understand and implement. However, there are substantial disadvantages: From a computational perspective, at least $n(d+1)$ simulations are required to estimate the gradient in all d coordinate directions; this can be a substantial burden if the optimization algorithm needs to repeatedly estimate the gradient as it searches. Also FD is biased because

$$\mathrm{E}\left(\mathrm{FD}(x_1)\right) = \frac{\theta(\mathbf{x}+\Delta x_1) - \theta(\mathbf{x})}{\Delta x_1},$$

while the derivative is the limit as $\Delta x_1 \to 0$. However, we cannot make Δx_1 too small because this causes numerical error as well as high variance since

$$\mathrm{Var}\left[(Y(\mathbf{x}+\Delta x_1) - Y(\mathbf{x}))/\Delta x_1\right] = \mathrm{Var}\left[Y(\mathbf{x}+\Delta x_1) - Y(\mathbf{x})\right]/\Delta x_1^2.$$

Common random numbers can provide substantial help with the variance problem and should virtually always be used (see Glasserman and Yao (1992)).

9.6.2 Infinitesimal Perturbation Analysis

Suppose that we fix the pseudorandom numbers at $(u_1, u_2, u_3, u_4, u_5)$. If either $-\ln(1-u_1)x_1 - \ln(1-u_4)x_4$ or $-\ln(1-u_1)x_1 - \ln(1-u_3)x_3 - \ln(1-u_5)x_5$ is the longest path, then it will also be on the longest path at $\mathbf{x} + \Delta x_1$ for Δx_1 small enough. And since nothing is random, the limit $\Delta x_1 \to 0$ of the FD estimator is easy to calculate:

$$\begin{aligned}
\lim_{\Delta x_1 \to 0} \frac{Y(\mathbf{x}+\Delta x_1) - Y(\mathbf{x})}{\Delta x_1} &= \lim_{\Delta x_1 \to 0} \frac{-\ln(1-u_1)(x_1 + \Delta x_1) - (-\ln(1-u_1)x_1)}{\Delta x_1} \\
&= -\ln(1-u_1) \\
&= \frac{-\ln(1-u_1)x_1}{x_1} = \frac{A_1(x_1)}{x_1}.
\end{aligned} \tag{9.21}$$

If neither of these is the longest path, then the difference does not depend on Δx_1 and

$$\lim_{\Delta x_1 \to 0} \frac{Y(\mathbf{x}+\Delta x_1) - Y(\mathbf{x})}{\Delta x_1} = 0. \tag{9.22}$$

These two cases together comprise the infinitesimal perturbation analysis (IPA) gradient estimator

$$\mathrm{IPA}(x_1) = \frac{A_1(x_1)}{x_1} I\{A_1(x_1) \text{ is on the longest path}\}. \tag{9.23}$$

Notice that $\mathrm{IPA}(x_1)$ requires no additional simulation, just an additional calculation.

Another heuristic argument that ends up at the same place is this: To estimate $\partial\theta(\mathbf{x})/\partial x_1$ it makes some sense to use $\partial Y(\mathbf{x})/\partial x_1$. If A_1 is on the longest path, then we can apply the chain rule to get

$$\begin{aligned}\frac{\partial Y(\mathbf{x})}{\partial x_1} &= \frac{\partial Y(\mathbf{x})}{\partial A_1} \times \frac{\partial A_1(x_1)}{\partial x_1} \\ &= 1 \times -\ln(1-U_1) = \frac{A_1(x_1)}{x_1}.\end{aligned}$$

If A_1 is not on the longest path, then the derivative is again 0.

We have not established that $\mathrm{IPA}(x_1)$ is a good gradient estimator (it is), but the reason it "works" is that, with probability 1, $Y(\mathbf{x})$ is continuous, differentiable, and bounded in a neighborhood of $\mathbf{x}$ for any fixed $\mathbf{x}$. Analogous IPA gradient estimators exist for x_2, x_3, x_4, x_5, and they can all be computed simultaneously from a single replication. Of course, to estimate the gradient we average $\mathrm{IPA}(x_1)$ over $n > 1$ replications.

More generally, the validity of IPA gradient estimation hinges on the validity of an interchange of differentiation and expectation:

$$\frac{\partial\theta(\mathbf{x})}{\partial x_i} = \frac{\partial \mathrm{E}[Y(\mathbf{x})]}{\partial x_i} \overset{?}{=} \mathrm{E}\left[\frac{\partial Y(\mathbf{x})}{\partial x_i}\right].$$

Technical conditions that allow this interchange are easy to state but often not as easy to verify for a simulation model.

9.6.3 Likelihood Ratio

One problem with FD is the need for $d+1$ simulations to estimate the full gradient. The following simple observation can get it down to 1.

Let $f_j(a_j|x_j) = \exp\{-a_j/x_j\}/x_j$ be the exponential distribution of the activity time A_j, $j = 1, 2, \ldots, 5$ for a fixed scenario $\mathbf{x}$. Thus, the joint distribution of the activity times is given by $\prod_{j=1}^{5} \exp\{-a_j/x_j\}/x_j$. Also let $g(\mathbf{a}) = \max\{a_1 + a_4, a_1 + a_3 +$

$a_5, a_2 + a_5\}$ where $\mathbf{a} = (a_1, a_2, a_3, a_4, a_5)$. Consider the expectation of a weighted version of the project completion time $Y(\mathbf{x})$:

$$\begin{aligned}
\mathrm{E}\left[Y(\mathbf{x})\frac{f_1(A_1|x_1+\Delta x_1)}{f_1(A_1|x_1)}\right] &= \mathrm{E}\left[Y(\mathbf{x})\frac{\exp\{-A_1/(x_1+\Delta x_1)\}/(x_1+\Delta x_1)}{\exp\{-A_1/x_1\}/x_1}\right] \\
&= \int_0^\infty \cdots \int_0^\infty g(\mathbf{a})\frac{f_1(a_1|x_1+\Delta x_1)}{f_1(a_1|x_1)} \prod_{j=1}^{5} f_j(a_j|x_j)\,\mathrm{d}\mathbf{a} \\
&= \int_0^\infty \cdots \int_0^\infty g(\mathbf{a}) f_1(a_1|x_1+\Delta x_1) \prod_{j=2}^{5} f_j(a_j|x_j)\,\mathrm{d}\mathbf{a} \\
&= \mathrm{E}[Y(\mathbf{x}+\Delta x_1)] = \theta(\mathbf{x}+\Delta x_1).
\end{aligned}$$

Notice that $A_1 \sim$ exponential with mean x_1, but by appropriately reweighting the output $Y(\mathbf{x})$ we can estimate $\theta(\mathbf{x}+\Delta x_1)$ from the observations generated under setting $\mathbf{x}$. The weight $f_1(A_1|x_1+\Delta x_1)/f_1(A_1|x_1)$ is sometimes called the likelihood ratio because it can be interpreted as the relative likelihood of the observation A_1 if the mean was $x_1 + \Delta x_1$ as compared to x_1. Therefore, from a single simulation replication we can get a finite-difference estimator:

$$\frac{Y(\mathbf{x})\dfrac{f_1(A_1|x_1+\Delta x_1)}{f_1(A_1|x_1)} - Y(\mathbf{x})}{\Delta x_1} = \frac{Y(\mathbf{x})}{f_1(A_1|x_1)} \times \frac{f_1(A_1|x_1+\Delta x_1) - f_1(A_1|x_1)}{\Delta x_1}.$$

As we did with FD to motivate IPA, we can now take the $\lim_{\Delta x_1 \to 0}$. Notice that

$$\lim_{\Delta x_1 \to 0} \frac{f_1(A_1|x_1+\Delta x_1) - f_1(A_1|x_1)}{\Delta x_1} = \frac{\partial f_1(A_1|x_1)}{\partial x_1}$$

by definition. Therefore, the likelihood ratio (LR) gradient estimator is

$$\mathrm{LR}(x_1) = \frac{Y(\mathbf{x})}{f_1(A_1|x_1)}\frac{\partial f_1(A_1|x_1)}{\partial x_1} = Y(\mathbf{x})\frac{\partial \ln f_1(A_1|x_1)}{\partial x_1}. \tag{9.24}$$

For the exponential distribution of activity 1

$$\frac{\partial \ln f_1(A_1|x_1)}{\partial x_1} = \frac{\partial}{\partial x_1}\left(-\frac{A_1}{x_1} - \ln x_1\right) = \frac{A_1}{x_1^2} - \frac{1}{x_1} = \frac{A_1 - x_1}{x_1^2}.$$

To estimate the gradient we average $\mathrm{LR}(x_1)$ over $n > 1$ replications. Clearly we can do the same thing for x_2, x_3, x_4, x_5. Again there are technical conditions that need to be satisfied for this to be a valid gradient estimator, and they are satisfied in this example. For LR to be applicable x must be the parameter of an input distribution.

9.6.4 Weak Derivative

We motivate this gradient estimator by wishful thinking. Recall that $\mathrm{FD}(x_1)$ comes from simulating A_1 from distribution f_1 with mean $x_1 + \Delta x_1$ and also with mean x_1, and then computing $(Y(\mathbf{x}+\Delta x_1) - Y(\mathbf{x}))/\Delta x_1$. $\mathrm{FD}(x_1)$ has expectation $(\theta(\mathbf{x}+\Delta x_1) - \theta(\mathbf{x}))/\Delta x_1$, which is not quite what we want. The wishful thinking is that there exist two other distributions for A_1, say $f_1^{(1)}$ and $f_1^{(2)}$, and a scaling constant $\Delta(x_1)$, that are just right, meaning

$$\mathrm{E}\left[\frac{g(A_1^{(1)},A_2,A_3,A_4,A_5) - g(A_1^{(2)},A_2,A_3,A_4,A_5)}{\Delta(x_1)}\right] = \frac{\partial \theta(\mathbf{x})}{\partial x_1}.$$

That is, we do two simulations, like FD, generating A_2,A_3,A_4,A_5 from their respective exponential distributions with means x_2,x_3,x_4,x_5, but we generate A_1 from $f_1^{(1)}$ for the first simulation, from $f_1^{(2)}$ for the second one, and then take a scaled finite difference.

Fortunately, it turns out that a magic triple $(\Delta(x), f^{(1)}, f^{(2)})$ exists for many common distributions (see Fu (2006), Table 1, noting that his $c = 1/\Delta(x)$). These are called weak derivatives (WD). For instance, for the exponential distribution with mean x a weak derivative is

$$\left(\Delta(x), f^{(1)}(a), f^{(2)}(a)\right) = \left(x, \frac{a}{x^2}\mathrm{e}^{-a/x}, \frac{1}{x}\mathrm{e}^{-a/x}\right).$$

Note that $a\mathrm{e}^{-a/x}/x^2$ is an Erlang-2 distribution with mean $2x$.

Think of WD as a FD gradient estimator with no bias. As with FD, common random numbers should be used for the two (or in general $2d$) simulations.

9.6.5 Linear Regression

To motivate the linear regression gradient estimator, suppose for the moment that $Y(\mathbf{x}) = \beta_0 + \beta^\top \mathbf{x} + \varepsilon$ where β_0 is a constant, β is $d \times 1$ vector of constants, and ε is a mean-zero, finite-variance random variable representing the stochastic simulation variability. Then clearly $\nabla\theta(\mathbf{x}) = \nabla\mathrm{E}(Y(\mathbf{x})) = \beta$. An unbiased estimator $\widehat{\beta}$ of β can be obtained by simulating n replications at each of $m \geq d+1$ design points $(\mathbf{x}_1, Y_j(\mathbf{x}_1)), (\mathbf{x}_2, Y_j(\mathbf{x}_2)), \ldots, (\mathbf{x}_m, Y_j(\mathbf{x}_m))$, $j = 1,2,\ldots,n$, and employing linear regression. We refer to this as the REG gradient estimator.

Of course for a complex stochastic simulation global linearity of $\mathrm{E}(Y(\mathbf{x}))$ as a function of $\mathbf{x}$ is probably too much to expect. However, near a fixed $\mathbf{x}_0$ approximate linearity through design points close to $\mathbf{x}_0$ might be plausible, and standard regression diagnostics can be applied to check this. Notice that the total simulation effort is $n \times m$ replications.

In the special case when $\mathbf{x}$ is a parameter of an input random variable, as it is in our SAN example, Wieland and Schmeiser (2006) observed that it may be possible to employ regression to estimate the entire gradient from n replications at $\mathbf{x}_0$, just as it is possible to do so with IPA and LR. Suppose that, in addition to $Y(\mathbf{x}_0)$, the simulation generates another $d \times 1$ random variable $\mathbf{X}$ that has two key properties:

1. $\mathrm{E}(\mathbf{X}) = \mathbf{x}_0$.
2. $\begin{pmatrix} Y(\mathbf{x}_0) \\ \mathbf{X} \end{pmatrix}$ has a multivariate normal distribution.

As a consequence of property 2, the conditional expected value of $Y(\mathbf{x}_0)$ given the observed value of $\mathbf{X}$ has the form $\mathrm{E}[Y(\mathbf{x}_0)|\mathbf{X}] = \beta_0 + \beta^\top \mathbf{X}$, and therefore

$$\mathrm{E}[Y(\mathbf{x}_0)] = \beta_0 + \beta^\top \mathrm{E}[\mathbf{X}] = \beta_0 + \beta^\top \mathbf{x}_0. \tag{9.25}$$

See Sect. 5.2.5.1 and specifically Eq. (5.10). Equation (9.25) implies that $\nabla \mathrm{E}(Y(\mathbf{x}_0)) = \beta$ and further that β can be estimated via linear regression of $Y_j(\mathbf{x}_0)$ on $\mathbf{X}_j,\ j = 1, 2, \ldots, n$.

Remark 9.5. Conditions 1 and 2 are sufficient but not necessary to employ the REG method of gradient estimation. If $\mathbf{X}$ is consistent for $\mathbf{x}_0$, and the conditional expectation $\mathrm{E}[Y(\mathbf{x}_0)|\mathbf{X}]$ is approximately linear then REG will tend to provide a useful estimate.

When might Conditions 1 and 2 hold at least approximately? In the SAN example $\mathbf{x}^\top = (x_1, x_2, \ldots, x_5)$ are the means of the activity times $\mathbf{X}^\top = (A_1, A_2, \ldots, A_5)$ so Condition 1 pertains. Clearly Condition 2 does not hold as the activity times are exponentially distributed. However, batching the replications $(Y_j(\mathbf{x}_0), \mathbf{X}_j),\ j = 1, 2, \ldots, n$, as described in Sect. 8.2.3 will improve the approximation of multivariate normality; batching can be a key tool to facilitate the regression method.

Wieland and Schmeiser (2006) focused on situations when a maximum likelihood estimator of $\mathbf{x}_0$ can be formed *within* each replication of the simulation from *many* observations. Consider for instance simulating the $M/G/1$ queue of Sect. 3.2 when the service times have a lognormal distribution with parameters $\mathbf{x}_0 = (\mu_0, \sigma_0^2)$. Each replication might generate hundreds to thousands of i.i.d. service times from which the MLE $\mathbf{X}^\top = (\widehat{\mu}, \widehat{\sigma}^2)$ can be computed *within each replication*. Then since MLEs are asymptotically normally distributed, Conditions 1 and 2 may be approximately satisfied.

9.6.6 Choice of Gradient Estimator

The appropriate gradient estimator depends very much on specifics of the simulation. The following discussion provides some guidance, but more detailed references like Fu (2006) and L'Ecuyer (1990) should be consulted.

FD is always applicable provided $\theta(\mathbf{x})$ is differentiable and $\widehat{\theta}(\mathbf{x})$ has finite variance in a neighborhood of $\mathbf{x}$. Of course, FD requires a minimum of $d+1$ simulations and a choice of Δx. Common random numbers should always be used. At the cost of $2d$ simulations a better estimator uses central differences

$$\mathrm{CD}(x_1) = \frac{Y(\mathbf{x}+\Delta x_1) - Y(\mathbf{x}-\Delta x_1)}{2\Delta x_1}.$$

If d is small or gradient estimates are only needed for a few scenarios $\mathbf{x}$, then FD and CD are attractive.

IPA, LR, and REG have the advantage of saving computational effort since they can be computed from a single simulation no matter what the dimension of $\mathbf{x}$. IPA tends to have smaller variance, but the conditions under which it is appropriate are sometimes difficult to verify and IPA can require more significant programming to implement (notice that in the SAN optimization IPA necessitated determining which activities were on the longest path). IPA gradient estimators are most often available when the simulation output itself can be represented—either directly or via the chain rule—as a differentiable function of x_1.

LR gradient estimators, on the other hand, are most often available when x_1 can be represented as a parameter of an input distribution. When LR gradient estimators exist, then computing them only requires reweighting the output of interest with a weight that is a function of the input distribution. LR gradient estimators tend to have larger variance than IPA, especially when x is the parameter of more than one random variate per replication because the weight is a product of the input distribution for each instance (e.g., x is the mean service time in a queue that simulates 1000 customers per replication so the weight will involve the product of the service time distribution 1000 times).

Like LR, single-run REG gradient estimators apply when $\mathbf{x}$ can be represented as the parameters of input distributions. Unlike LR, REG is most appropriate when more than one random variate with $\mathbf{x}$ as its parameter is generated per replication. REG estimators can be low variance and employ only ubiquitious linear regression software to compute. However, they require retaining the values of the input random variates generated during the replications.

Like LR and REG, WD gradient estimators can be considered when x is a parameter of an input distribution. WD gradient estimators do not save simulations relative to FD and will be most useful when the input whose distribution has parameter $\mathbf{x}$ appears only once per replication, as it did in the SAN simulation. WD gradient estimators tend to have low variance when common random numbers are employed.

Just like the simulation response estimator $\widehat{\theta}(\mathbf{x}; T, n, \mathbf{U})$ itself, the variance of the gradient estimator depends on the number of replications and, for a steady-state simulation, the run length. The number of replications or run length required to get a precise estimate of $\theta(\mathbf{x})$ may or may not be sufficient to obtain a precise estimate of its gradient.

9.6.7 Steepest Descent

Gradient-based search for SO is beyond the scope of this book, other than to describe the simplest method known as *stochastic approximation*,[3] which is a method of steepest descent: Starting from an initial scenario $\mathbf{x}_0$, implement the recursion

$$\mathbf{x}_{i+1} = \mathbf{x}_i - a_i \widehat{\nabla}\theta(\mathbf{x}_i), \tag{9.26}$$

where a_i is a sequence such that $a_i \to 0$, but $\sum_{i=1}^{\infty} a_i = \infty$; setting $a_i = a/i$ for a positive constant a is a common choice. The intuition is that when $\nabla\theta(\mathbf{x}_i) = 0$ a stationary point has been reached that is at least a local minimum, while having $a_i \to 0$ (but not too fast) mutes the impact of variability in the gradient estimator $\widehat{\nabla}\theta(\mathbf{x}_i)$.

Stochastic approximation can be sensitive to the choice of sequence a_i, so it is often made adaptive. A second issue is that on some iterations $\mathbf{x}_{i+1}$ may end up outside the feasible region C; then there must be some way to project it back into C.

For instance, the constraints for the SAN optimization are

$$\begin{aligned}&\sum_{j=1}^{5} c_j(\tau_j - x_j) \le b\\ &x_j \ge \ell_j,\ j = 1,2,3,4,5.\end{aligned}$$

If $\sum_{j=1}^{5} c_j(\tau_j - x_{i+1,j}) > b$, then we might instead use $\mathbf{x}'_{i+1}$, which is the perpendicular projection of $\mathbf{x}_{i+1}$ onto the plane $\sum_{j=1}^{5} c_j(\tau_j - x_j) = b$, provided it also satisfies the lower bound constraints. Exercise 5 asks you to work this out.

9.7 Optimizing the Sample Average Problem

In SO our goal is to minimize $\theta(\mathbf{x})$ using a simulation-based estimator $\widehat{\theta}(\mathbf{x};T,n,\mathbf{U})$ of the objective function; we will assume that the replication stopping time T is not a factor and drop it from the notation. Given fixed values for the number of replications n and, most importantly, the random numbers $\mathbf{U} = \mathbf{u}$, consider solving instead the optimization problem

$$\begin{aligned}&\min\ \widehat{\theta}(\mathbf{x};n,\mathbf{U}=\mathbf{u})\\ &\mathbf{x} \in \mathsf{C}.\end{aligned} \tag{9.27}$$

In words, the objective of the optimization problem in (9.27) is to minimize the value of the *estimator* $\widehat{\theta}(\mathbf{x};n,\mathbf{U}=\mathbf{u})$ as a function of $\mathbf{x}$. With the random numbers fixed, this is a deterministic optimization problem and it seems plausible that if n

[3] The name *stochastic gradient descent* is another name for this method, particularly when applied in a machine learning context.

is large enough then the optimal solution to (9.27) should be close to the optimal solution of the SO problem. This approach is called *sample average approximation (SAA).*

For instance, the output from replication j of the SAN simulation can be expressed as

$$\begin{aligned}
y_j(\mathbf{x}) &= \max\{a_{1j}(x_1)+a_{4j}(x_4),\\
&\qquad a_{1j}(x_1)+a_{3j}(x_3)+a_{5j}(x_5),\\
&\qquad a_{2j}(x_2)+a_{5j}(x_5)\}\\
&= \max\{-\ln(1-u_{1j})x_1-\ln(1-u_{4j})x_4,\\
&\qquad -\ln(1-u_{1j})x_1-\ln(1-u_{3j})x_3-\ln(1-u_{5j})x_5,\\
&\qquad -\ln(1-u_{2j})x_2-\ln(1-u_{5j})x_5\},
\end{aligned}$$

where we have made Y, A and U lowercase to indicate that there are no random variables left after the random numbers are fixed. The objective of the corresponding SAA problem is

$$\min\ \widehat{\theta}(\mathbf{x};n,\mathbf{U}=\mathbf{u}) = \min \frac{1}{n}\sum_{j=1}^{n} y_j(\mathbf{x}).$$

In many cases the SAA problem is a difficult nonlinear optimization problem without apparent structure and therefore cannot be readily solved. But in some SO problems, such as the SAN optimization (9.2), solving the SAA problem is easy. In fact, if we replace the max operator by three inequality constraints, then the SAA problem can be formulated as a linear program:

$$\begin{aligned}
\min\quad & \frac{1}{n}\sum_{j=1}^{n} y_j \qquad\qquad (9.28)\\
& y_j \ge -\ln(1-u_{1j})x_1-\ln(1-u_{4j})x_4\\
& y_j \ge -\ln(1-u_{1j})x_1-\ln(1-u_{3j})x_3-\ln(1-u_{5j})x_5\\
& y_j \ge -\ln(1-u_{2j})x_2-\ln(1-u_{5j})x_5,\ j=1,2,\ldots,n\\
& b \ge \sum_{k=1}^{5} c_k(\tau_k - x_k)\\
& x_k \ge \ell_k,\ k=1,2,\ldots,5.
\end{aligned}$$

This linear program has $n+5$ decision variables $y_1,y_2,\ldots,y_n,x_1,x_2,\ldots,x_5$ and $3n+6$ constraints (remember that all of the $-\ln(1-u_{ij})$ terms are fixed constants and therefore are coefficients in the linear program). Thus, the SAA problem can be solved even for a very large number of replications, n.

Let $\widehat{\mathbf{x}}_n^\star$ be the optimal solution to the SAA problem (9.27) based on n replications. Then what matters is whether $\theta(\widehat{\mathbf{x}}_n^\star)$ is close to $\theta(\mathbf{x}^\star)$, and in particular does $\theta(\widehat{\mathbf{x}}_n^\star) \to \theta(\mathbf{x}^\star)$ as $n\to\infty$? The various conditions that guarantee convergence are quite technical, so we present just one set here. A comprehensive reference is Shapiro et al. (2009).

Suppose that for any $\mathbf{x} \in \mathsf{C}$ we have $\widehat{\theta}(\mathbf{x}; n, \mathbf{U}) \to \theta(\mathbf{x})$ with probability 1 as the number of replications $n \to \infty$; that is, $\widehat{\theta}(\mathbf{x}; n, \mathbf{U})$ satisfies a strong law of large numbers as is often the case for averages across replications (see Sect. 5.2.2). From here on we will assume that $\widehat{\theta}(\mathbf{x}; n, \mathbf{U})$ is an average across i.i.d. replications and $\mathrm{E}[\widehat{\theta}(\mathbf{x}; n, \mathbf{U})] = \theta(\mathbf{x})$.

Although pointwise convergence might seem like enough, it is insufficient for convergence of the optimal value $\theta(\widehat{\mathbf{x}}_n^\star) \to \theta(\mathbf{x}^\star)$ in general because $\widehat{\mathbf{x}}_n^\star$ is not a fixed value of $\mathbf{x}$, but rather the solution to the SAA problem over *all* $\mathbf{x} \in \mathsf{C}$. A more comprehensive form of convergence is needed, in particular $\widehat{\theta}(\mathbf{x}; n, \mathbf{U}) \to \theta(\mathbf{x})$ *uniformly* for all $\mathbf{x} \in \mathsf{C}$.

Uniform convergence means that, for any $\varepsilon > 0$, with probability 1, there exists an n' such that

$$\sup_{\mathbf{x} \in \mathsf{C}} \left| \widehat{\theta}(\mathbf{x}; n; \mathbf{U}) - \theta(\mathbf{x}) \right| \le \varepsilon$$

for all $n \ge n'$. For pointwise convergence this n' can depend on $\mathbf{x}$; for uniform convergence it has to be independent of $\mathbf{x}$, although it can depend on the particular sequence of random numbers $\mathbf{U}$. What it means is that when the number of replications is large enough, the estimator $\widehat{\theta}(\mathbf{x}; n; \mathbf{U})$ is uniformly close to $\theta(\mathbf{x})$ for all $\mathbf{x}$. As a practical matter, if the estimator $\widehat{\theta}(\mathbf{x}; n; \mathbf{U})$ is continuous in $\mathbf{x}$ with probability 1, and $\theta(\mathbf{x})$ is bounded, then we will have uniform convergence. Verifying this has to be done problem by problem.

When it applies, SAA can be a powerful SO method because it turns a stochastic problem into a deterministic one to which we can apply sophisticated linear and nonlinear optimization techniques. A drawback is that the computational requirements to solve the SAA problem can increase substantially in n, and we want n large to obtain a good approximation.

9.8 Stochastic Constraints

Recall again the formulation of the simulation optimization (SO) problem that opened this section:

$$\begin{aligned} &\min\ \theta(\mathbf{x}) \\ &\mathbf{x} \in \mathsf{C}\,. \end{aligned}$$

What makes it a SO problem is the need to estimate $\theta(\mathbf{x})$ using a stochastic simulation. We have assumed that for any scenario $\mathbf{x}$ we can determine with certainty whether $\mathbf{x}$ is feasible ($\mathbf{x} \in \mathsf{C}$) or infeasible ($\mathbf{x} \notin \mathsf{C}$). For many practical problems this is not possible. For instance, in the service center simulation of Sect. 4.6, the objective was to minimize staffing costs subject to two constraints:

Percentage of simple orders entered in $\le$10 min is at least 96%.
Percentage of special orders entered in $\le$10 min is at least 80%.

Given a staffing scenario $\mathbf{x}$, satisfaction of these constraints has to be estimated.

For the service center the objective function (minimize staffing cost) was deterministic, but more generally we have to estimate it as well. So we expand the SO formulation to include stochastic constraints, specifically

$$\begin{aligned} &\min\ \theta(\mathbf{x}) && (9.29)\\ &\mathbf{x} \in \mathsf{C} \\ &c_\ell(\mathbf{x}) \ge q_\ell,\ \ell = 1,2,\ldots,v, && (9.30) \end{aligned}$$

where $\theta(\mathbf{x}), c_1(\mathbf{x}), c_2(\mathbf{x}), \ldots, c_v(\mathbf{x})$ must all be estimated. For the purposes of this section we assume that for any scenario $\mathbf{x}$ we can observe simulation outputs $Y(\mathbf{x}), C^{(1)}(\mathbf{x}), C^{(2)}(\mathbf{x}), \ldots, C^{(v)}(\mathbf{x})$ with the property that

$$\begin{aligned} \mathrm{E}(Y(\mathbf{x})) &= \theta(\mathbf{x}) \\ \mathrm{E}\left(C^{(\ell)}(\mathbf{x})\right) &= c_\ell(\mathbf{x}),\ \ell = 1,2,\ldots,v \end{aligned}$$

so that it makes sense to estimate $\theta(\mathbf{x})$ and $c_\ell(\mathbf{x})$ by sample averages.

For example, in the service center simulation let $C^{(1)}(\mathbf{x})$ be the observed fraction of simple orders arriving between 8 a.m. and 4 p.m. and that are entered in $\le$10min, and let $C^{(2)}(\mathbf{x})$ be the corresponding fraction for special orders, under staffing scenario $\mathbf{x}$. Then the stochastic constraints are

$$\begin{aligned} c_1(\mathbf{x}) = \mathrm{E}\left(C^{(1)}(\mathbf{x})\right) &\ge 0.96 \\ c_2(\mathbf{x}) = \mathrm{E}\left(C^{(2)}(\mathbf{x})\right) &\ge 0.80. \end{aligned}$$

To simplify the presentation we will focus on the case of $v = 1$ stochastic constraint $c(\mathbf{x}) \ge q$ that is estimated by the simulation output $C(\mathbf{x})$.

Constraints nearly always represent limits on our ability to minimize the objective $\theta(\mathbf{x})$; stated differently, if the constraints could be relaxed or removed, then scenarios could be chosen to make $\theta(\mathbf{x})$ even smaller. In the service center we could certainly assign fewer staff if, say, only 50% of the orders needed to be entered in 10 min or less. Therefore, at the optimal scenario $\mathbf{x}^\star$ we expect the constraints to be tight ($c(\mathbf{x}^\star) = q$) or nearly so. *A key message of this section is that using simulation-based estimates to decide if $c(\mathbf{x}) \ge q$ when the constraint is tight, or nearly so, is a more difficult problem than estimating the value of $\theta(\mathbf{x})$.*

We will not recommend specific algorithms for dealing with stochastic constraints; instead, we make clear the difficulties that arise when including them. This section is somewhat advanced and can be skipped without loss of continuity.

9.8.1 Feasibility

When there are a relatively small number of scenarios to consider, then a reasonable strategy is to first decide which of them are feasible and next conduct a simulation experiment to find the best among the feasible ones. How hard is it to decide if a scenario $\mathbf{x}$ is feasible?

Suppose that for scenario $\mathbf{x}$ we run n i.i.d. replications to obtain observations of the constrained output $C_1(\mathbf{x}), C_2(\mathbf{x}), \ldots, C_n(\mathbf{x})$. In the service center simulation this might be the observed fraction of simple orders entered in ≤ 10 min for staffing level $\mathbf{x}$ from n replications (days). We then decide constraint satisfaction by checking

$$\overline{C}(\mathbf{x}) \overset{?}{\geq} q.$$

Let $\sigma^2(\mathbf{x}) = \mathrm{Var}\left(C(\mathbf{x})\right)$ and assume the usual case that $0 < \sigma^2(\mathbf{x}) < \infty$. Then by the central limit theorem (Sect. 5.2.2)

$$\begin{aligned}\Pr\{\overline{C}(\mathbf{x}) \geq q\} &= \Pr\left\{\frac{\sqrt{n}\left(\overline{C}(\mathbf{x}) - c(\mathbf{x})\right)}{\sigma(\mathbf{x})} \geq \frac{\sqrt{n}\,(q - c(\mathbf{x}))}{\sigma(\mathbf{x})}\right\} \\ &\overset{n\to\infty}{\longrightarrow} \Pr\{\mathrm{N}(0,1) \geq \lambda\}, \end{aligned} \tag{9.31}$$

where $\mathrm{N}(0,1)$ represents a standard normal random variable. There are three cases for λ:

1. If $\mathbf{x}$ is infeasible, then $q - c(\mathbf{x}) > 0$, so λ is ∞ and (9.31) is 0. This is what we want.
2. If $\mathbf{x}$ is feasible but not tight, then $q - c(\mathbf{x}) < 0$, so λ is $-\infty$ and (9.31) is 1. This is also what we want.
3. If $\mathbf{x}$ is tight, then $q - c(\mathbf{x}) = 0$, so λ is 0 and (9.31) is $1/2$. *This means that no matter how much simulation we do, we cannot determine with certainty if a constraint is tight, which implies that it will also be difficult when the constraint is close to being tight.*

When checking feasibility of stochastic constraints, there are two types of errors: declaring a scenario feasible when it is infeasible, and declaring a scenario infeasible when it is feasible. Often, however, the seriousness of the error depends on the difference $q - c(x)$. In the service center if we declare that a staffing policy $\mathbf{x}$ is feasible when in fact $c_1(\mathbf{x}) = 0.77$, then this is certainly a more serious error than declaring $\mathbf{x}'$ feasible when $c_1(\mathbf{x}') = 0.94$.

A practical way to address this difficulty is to expand the concept of feasibility. Choose two additional cut-offs, q^- and q^+ such that $q^- < q < q^+$. If the value of $c(\mathbf{x}) \geq q^+$, then scenario $\mathbf{x}$ is *desirable*; if $c(\mathbf{x}) \leq q^-$, then scenario $\mathbf{x}$ is *unacceptable*, while if $q^- < c(\mathbf{x}) < q^+$, then secenario $\mathbf{x}$ is *acceptable*. We then run experiments that allow us to be confident that the scenarios we declare feasible include all of the desireable ones and some or all of the acceptable ones, but does not include unacceptable scenarios. By allowing the slack to declare acceptable scenarios to be feasible—whether or not they are actually feasible—we avoid the difficulty with tight constraints.

Andradóttir and Kim (2010) and Batur and Kim (2010) provide formal procedures to accomplish this. Informally, we can estimate $c(\mathbf{x})$ precisely enough, as measured by a confidence interval $\overline{C}(\mathbf{x}) \pm H$, so that we can feel confident that $\mathbf{x}$

is in the desirable to acceptable range (we do not have to separate them), or in the unacceptable range; we retain the former and discard the latter.

Returning to the service center example, if the requirement of at least 96% of all simple orders being entered in ≤ 10 min is not a firm requirement, then we might set $q^- = 0.92$ and $q^+ = 0.98$, for instance. This means that we want to be highly confident of declaring staffing scenarios that achieve 98% entry as feasible, to discard scenarios that are worse than 92%, and are willing to consider scenarios that achieve greater than a 92% but necessarily a 96% service level.

9.8.2 Constraint-Guided Search

Here we consider SO problems that require a search, rather than exhausting all possible scenarios. One approach is to represent violation of the stochastic constraints as a penalty. Two cases in this setting are particularly important.

Suppose that it is physically possible to violate the constraints, but the greater the violation the less desirable the scenario is. This is certainly true in the service center example. We might then reformulate the SO problem as

$$\begin{aligned} &\min\ \theta(\mathbf{x}) + \sum_{\ell=1}^{v} p_\ell\left(q_\ell - c_\ell(\mathbf{x})\right) \\ &\mathbf{x} \in \mathsf{C}, \end{aligned} \tag{9.32}$$

where $p_\ell(\cdot)$ is a penalty function that is 0 if $c_\ell(\mathbf{x}) \geq q_\ell$ and increasing in $q_\ell - c_\ell(\mathbf{x})$ otherwise. For instance,

$$p_\ell\left(q_\ell - c_\ell(\mathbf{x})\right) = \beta_\ell \max\left\{0, q_\ell - c_\ell(\mathbf{x})\right\}.$$

This converts the SO problem (9.32) with stochastic constraints into a SO problem with deterministic constraints to which the methods described elsewhere in this section can be applied. Clearly the choice of the penalty function is important; in the service center problem it should convert underachieving the service level into a dollar cost (since the primary objective is staffing cost). Beyond that, there is a subtle statistical issue that should be noted.

Returning to the case of $v = 1$ constraint, suppose we have n replications of the simulation at scenario $\mathbf{x}$. How do we estimate the Objective (9.32)? A natural (and valid) estimator is

$$\bar{Y}(\mathbf{x}) + p\left(q - \bar{C}(\mathbf{x})\right). \tag{9.33}$$

If the penalty function $p(\cdot)$ is continuous around $q - c(\mathbf{x})$, then this estimator converges to the desired objective $\theta(\mathbf{x}) + p\left(q - c(\mathbf{x})\right)$ as $n \to \infty$ by the strong law of large numbers and the continuous mapping theorem (see Sect. 5.2.4).

However, one might be tempted to compute the penalty on each replication and use

$$\frac{1}{n}\sum_{j=1}^{n}\left[Y_j(\mathbf{x})+p\left(q-C_j(\mathbf{x})\right)\right]. \tag{9.34}$$

This estimator will converge to

$$\mathrm{E}\left[Y(\mathbf{x})+p\left(q-C(\mathbf{x})\right)\right]\neq\theta(\mathbf{x})+p(q-c(\mathbf{x}))$$

unless $p(\cdot)$ is linear. And since the penalty function is nonnegative, even a strictly feasible scenario may have a positive penalty in the limit.

Now suppose that we really want to enforce feasibility, at least asymptotically. Then instead of penalizing the objective function as in (9.32) we penalize the estimated value of the scenario. To describe this approach, recall that search algorithms work iteratively attempting to find better and better scenarios as the number of iterations increases. Let $i=0,1,2,\ldots$ be the iteration index, and let $n_i(\mathbf{x})$ be the total number of replications obtained from scenario $\mathbf{x}$ through iteration i (search algorithms can and often do revisit scenarios). Then in the case of a single constraint we might define the estimated value of scenario $\mathbf{x}$ through iteration i to be

$$\widehat{\theta}(\mathbf{x})=\overline{Y}(\mathbf{x})+\beta_i\max\left\{0,q-\overline{C}(\mathbf{x})\right\},$$

where the averages include all observations $j=1,2,\ldots,n_i(\mathbf{x})$. Notice that the coefficient β_i depends on the iteration i (it might also depend on $n_i(\mathbf{x})$) because we want the penalty to grow as the search progresses so that, at least asymptotically, infeasible scenarios are not competitive. The point to be made here is that one needs to be careful about using the simple prescription that $\beta_i\to\infty$ as $i\to\infty$. Here is why:

Suppose that the number of replications at scenario $\mathbf{x}$, $n_i(\mathbf{x})$, goes to infinity as the number of iterations goes to infinity. What happens to the penalty $\beta_i\max\left\{0,q-\overline{C}(\mathbf{x})\right\}$? It makes intuitive sense and can be proven formally using the strong law of large numbers (see Sect. 5.2.2) that if $\mathbf{x}$ is infeasible, then $\beta_i\max\left\{0,q-\overline{C}(\mathbf{x})\right\}\xrightarrow{a.s.}\infty$, while if $\mathbf{x}$ is feasible but not tight then $\beta_i\max\left\{0,q-\overline{C}(\mathbf{x})\right\}\xrightarrow{a.s.}0$, even though $\beta_i\to\infty$.

But this is misleading. Consider the probability of a penalty larger than, say, $\delta>0$:

$$\begin{aligned}
&\Pr\left\{\beta_i\max\left\{0,q-\overline{C}(\mathbf{x})\right\}>\delta\right\}\\
&\geq\Pr\left\{\beta_i\left(q-\overline{C}(\mathbf{x})\right)>\delta\right\}\\
&=\Pr\left\{\overline{C}(\mathbf{x})-q<-\frac{\delta}{\beta_i}\right\}\\
&=\Pr\left\{\frac{\sqrt{n_i(\mathbf{x})}\left(\overline{C}(\mathbf{x})-c(\mathbf{x})\right)}{\sigma}<\frac{\sqrt{n_i(\mathbf{x})}}{\sigma}\left(q-c(\mathbf{x})-\frac{\delta}{\beta_i}\right)\right\}.
\end{aligned}$$

Using a central limit theorem argument similar to Sect. 9.8.1 the random variable on the left-hand side is asymptotically $\mathrm{N}(0,1)$. Consider three cases for the right-hand side:

1. If $\mathbf{x}$ is infeasible, then we want this probability to get large quickly. Since $q - c(\mathbf{x}) > 0$ and $\delta > 0$, this means we would like β_i to become large quickly (big penalty), so $\beta_i \to \infty$ is fine.
2. If $\mathbf{x}$ is feasible but not tight, then we want this probability to get small quickly (no penalty). Since $q - c(\mathbf{x}) < 0$ and $\delta > 0$, we would like β_i to remain small and ideally go to 0.
3. If $\mathbf{x}$ is tight, then we also want this probability to get small quickly (no penalty). Since $q - c(\mathbf{x}) = 0$, we *need* β_i to go to 0 to avoid a penalty.

Thus, contrary to the simple prescription, we actually want $\beta_i \to \infty$ only for infeasible scenarios (whose identities we do not know in advance), suggesting that penalties should adapt to the observed (in)feasibility.

9.9 Parallelizing Simulation Optimization

The potential benefit of executing portions of the SO algorithms described in Sects. 9.3–9.8 in parallel is obvious, but—as described in Sect. 5.3—achieving the full potential of parallel simulation, including statistically valid inference, is not straightforward. To date the greatest successes in parallel SO with statistical guarantees have come in the ranking-and-selection setting (Sect. 9.3), but the discussion here applies more generally.[4]

To avoid the need to specify any particular computer architecture we consider a "processor" to be a computational engine capable of executing one "job" at a time; therefore, a parallel computer system is one that can execute more than one job simultaneously. As a working representation we use the master–worker architecture illustrated in Fig. 5.6 consisting of $p+1$ processors with all communication going through the master. Worker to worker communication may also make sense and we do not prohibit it.

We define job j as an ordered list that specifies the simulation replications to obtain, if any, and non-simulation calculations to perform, if any:

$$\mathsf{Job}_j = \{(\mathscr{X}_j, \mathscr{N}_j, \mathbf{U}_j), (\mathscr{P}_j, \mathscr{C}_j)\},$$

where

- $\mathscr{X}_j = \{\mathbf{x}_{ij}\}$ is a set of scenarios to be simulated;
- $\mathscr{N}_j = \{n_{\mathbf{x}_{ij}}\}$ is the number of replications to obtain from each scenario in $\mathscr{X}_j$;
- $\mathbf{U}_j$ is the block of pseudorandom numbers assigned to the replications (for assigning pseudorandom numbers in parallel see Sect. 6.5.3);

[4] This section is based on Hunter and Nelson (2017).

- $\mathscr{P}_j$ is a list of jobs whose completion must precede the calculations $\mathscr{C}_j$; and
- $\mathscr{C}_j$ is a list of non-simulation calculations to perform.

Some jobs may only specify simulation replications, others may only specify non-simulation calculations, and yet others may specify both. The execution of a job requires wall-clock time, which may include set-up time (e.g., the time for communication from the master to a worker) as well as execution time for replications or calculations. By "wall-clock time" we mean the time required to complete the SO, not the clock that is internal to the simulation.

From this perspective, a SO algorithm creates a sequence of jobs to be executed, $\mathscr{J} = \{\mathsf{Job}_j\colon j = 1,2,\ldots,M\}$, where M may be random or even infinite in an algorithm without a specific stopping condition. A *parallel* SO algorithm executes more than one job at a time, which may reduce wall-clock time to complete $\mathscr{J}$ but may (and likely will) require synchronization of jobs so that the requirements of $(\mathscr{P}_j,\mathscr{C}_j)$ are respected (i.e., so that non-simulation calculations have access to the simulation replication results that they need from preceding jobs).

As a simple illustration, consider the subset selection procedure of Sect. 9.3.1 when common random numbers are employed and we obtain n replications from each system. This algorithm can be represented as K simulation jobs, followed by one calculation job to compute the pairwise comparisons and return the subset I; we indicate whether a worker or the master performs each job below:

Subset Selection

$$
\begin{array}{ll}
\mathsf{Job}_1 = \{(\mathbf{x}_1,n,\mathbf{U}),(\emptyset,\emptyset)\} & \text{worker}\\
\mathsf{Job}_2 = \{(\mathbf{x}_2,n,\mathbf{U}),(\emptyset,\emptyset)\} & \text{worker}\\
\quad\vdots & \\
\mathsf{Job}_k = \{(\mathbf{x}_K,n,\mathbf{U}),(\emptyset,\emptyset)\} & \text{worker}\\
\mathsf{Job}_{K+1} = \{(\emptyset,\emptyset,\emptyset),(\{\mathsf{Job}_1,\mathsf{Job}_2,\ldots,\mathsf{Job}_K\},\{\text{Steps 2–3 of subset}\})\}. & \text{master}
\end{array}
$$

In parallel, simulation jobs are assigned to idle workers until $\mathsf{Job}_1,\mathsf{Job}_2,\ldots,\mathsf{Job}_K$ are all completed; then the master executes the single computation Job_{K+1} to compute pairwise comparisons and return the surviving subset. Notice that there is only one job that requires synchronization, the last one. If K and p are large, then the speed-up will be substantial: in fact, except for the last job the speed-up is linear in the number of workers, p. Of course, the workers are entirely idle, while the master executes Job_{K+1}, which may take significant wall-clock time if K is large.

Unfortunately, SO algorithms often gain *replication efficiency* by employing synchronization steps that hinder *computational efficiency* when executed in parallel. Consider the selection-of-the-best algorithm of Sect. 9.3.2. After simulating all K scenarios for an initial n_0 replication, this algorithm is essentially repeated iteration of the subset-selection jobs above until the size of the subset is 1 (see Steps 3–4 of select the best).

Specifically, on each iteration, the workers execute $|I|$ simulation jobs—one for each surviving scenario—of one replication each, followed by a pairwise-comparison job on the master when all $|I|$ simulation jobs have completed. Notice that every time the master undertakes a comparison job, the p workers are idle until the new set I of survivors is determined. Further, both the master and many of the workers may be idle waiting for the last few simulation jobs to complete on each iteration. When $|I| < p$, some workers are always idle. Finally, each replication from each scenario requires set-up time for communication between the master and worker (i.e., the scenario to simulate and the result to return), which in total may take significant wall-clock time. All of this is computationally inefficient.

Parallel SO is a rapidly evolving research area, with the generic goals of being fast, cheap, and effective. In an attempt to provide a more precise formulation of the goals, Hunter and Nelson (2017) defined fixed-precision and fixed-budget formulations of ranking and selection, but the concepts apply to SO more generally. Their formulations recognize that there is often a cost to purchase parallel computing time, and therefore the number of parallel processors to rent and for how long become part of the SO algorithm.

Define the following additional quantities:

- $T(\mathscr{J})$ is the wall-clock time (possibly random) when a SO algorithm specified by the jobs $\mathscr{J}$ terminates.
- $c(p,s)$ is the cost to purchase p processors for s time units.
- G is some desirable "good event," such as correct selection of the optimal scenario, and B is some "bad event," such as choosing a scenario whose expected value is unacceptably far from that of the optimal scenario.

One possible goal for a fixed-precision SO algorithm is

$$\min_{p,\mathscr{J}} \mathrm{E}\left[\beta_t T(\mathscr{J}) + \beta_c c(p, T(\mathscr{J}))\right] \quad \text{s.t.} \quad \Pr\{G\} \ge 1-\alpha.$$

In words, choose the jobs $\mathscr{J}$ and the number of parallel processors p to minimize the expected value of a weighted average of the wall-clock time to complete the optimization and the cost to purchase the computer time, subject to a guarantee on the good event occurring. Typically one of β_t or β_c is 0. One possible goal for a fixed-budget SO algorithm is

$$\min_{p,\mathscr{J}} \mathrm{E}\left[\mathrm{Loss}(B,\mathscr{J})\right] \quad \text{s.t.} \quad c(p,T(\mathscr{J})) \le b.$$

Here "Loss" is some measure of regret if the bad event occurs, such as the size of the optimality gap when an inferior scenario is chosen. While no SO algorithm of which we are aware directly attacks these goals, they emphasize that there is an interplay between the algorithm and the processor capacity that all parallel SO algorithms must address in some way.

Remark 9.6. Notice that we have not discussed obtaining replications from *the same scenario* $\mathbf{x}$ *in parallel from multiple processors* due to the bias issues described in Sect. 5.3. However, there will be settings in which parallelizing replications of the same scenario is advantageous, and by delaying computations $\mathscr{C}$ until all specified simulation jobs $\mathscr{P}$ are completed the bias can be avoided at the cost of idling processors.

Remark 9.7. To the best of our knowledge there are not yet comprehensive references on parallel simulation optimization, although there are a number of papers on parallel ranking and selection. Pei et al. (2020) provide an entry into that literature, as well as discussing and evalutating the computational issues using parallel subset selection as the baseline for comparison.

9.10 Sensitivity Analysis

Section 7.2.3 introduced the concept of sensitivity analysis for the properties of a simulation output $Y(\mathbf{x})$ with respect to some decision variables or parameters $\mathbf{x}$, contrasting it with input uncertainty. In this section we use the term "sensitivity analysis" to mean evaluating what happens to $Y(\mathbf{x})$ when $\mathbf{x}$ is perturbed locally from some nominal value $\mathbf{x}_0$.

Let the generic argument be $\mathbf{x}^\top = (x_1, x_2, \ldots, x_d)$, with $\mathbf{x}_0^\top = (x_{10}, x_{20}, \ldots, x_{d0})$ the nominal setting, and $\theta(\mathbf{x}) = \mathrm{E}(Y(\mathbf{x}))$. If $\theta(\mathbf{x})$ is differentiable in $\mathbf{x}$, then the gradient at $\mathbf{x}_0$,

$$\nabla\theta(\mathbf{x}_0)^\top = \left(\frac{\partial\theta(\mathbf{x})}{\partial x_1}, \frac{\partial\theta(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial\theta(\mathbf{x})}{\partial x_d}\right)\Bigg|_{\mathbf{x}=\mathbf{x}_0},$$

is a local sensitivity measure. Therefore, the gradient estimation methods described in Sect. 9.6 can be exploited to estimate it.

To illustrate, recall that in the SAN example where $\mathbf{x}^\top = (x_1, x_2, \ldots, x_5)$ are the means of the exponentially distributed activity times $(A(x_1), A(x_2), \ldots, A(x_5))$, and $Y(\mathbf{x})$ is the time to complete the network. Thus, the $\partial\theta(\mathbf{x}_0)/\partial x_i$ is the increase in the expected value of the time to complete the project per unit increase in the mean of activity i. Or, if our goal is to speed up the project, then $-\partial\theta(\mathbf{x}_0)/\partial x_i$ is the decrease in the expected value of the time to complete the project per unit decrease in the mean of activity i.

When $\mathbf{x}$ refers to input model parameters, as in the SAN example, interpreting the gradient can be a problem when the distribution has more than one parameter. For instance, suppose that activity 1 has a gamma distribution with parameters $\mathbf{x}^\top = (x_1, x_2) = (\alpha, \beta)$. Alhough we can estimate the gradient of $\mathrm{E}(Y(\mathbf{x}_0))$ with respect to the shape and scale parameters, they are less meaningful than sensitivity with respect to the *mean* of activity 1. Unfortunately, the mean of activity 1 is $x_1/x_2 = \alpha/\beta$, so there are infinitely many ways that (x_1, x_2) can change to increase the mean by one unit.[5] Which one do we want?

Jiang et al. (2019) suggest selecting an interpretable *direction* for the change in the input parameters $\mathbf{x}$, and then using directional derivatives to estimate the change in the output property as the input changes along the chosen direction. For instance, if we represent the mean of the gamma distribution as $\mu(x_1, x_2) = x_1/x_2$, then the

[5] This is one of two common ways to parameterize the gamma distribution.

steepest ascent direction—that is, the direction along which the mean of the gamma distribution changes the fastest—is the gradient direction

$$\mathbf{d}^\top = \nabla\mu(x_1,x_2)^\top = \left(\frac{1}{x_2}, -\frac{x_1}{x_2^2}\right).$$

This direction can be interpreted as the best or worst case, depending on the context.

Jiang et al. (2019) propose an interpretable sensitivity measure

$$\frac{\mathbf{d}^\top \nabla\theta(x_1,x_2)}{\mathbf{d}^\top \nabla\mu(x_1,x_2)}, \tag{9.35}$$

which is the ratio of the rate of change of $\theta(\mathbf{x})$ with respect to $\mathbf{x}$ relative to the rate of change of $\mu(x_1,x_2)$ with respect to $\mathbf{x}$, as $\mathbf{x}$ moves along the direction in which the mean changes the fastest. The numerator can be estimated using the methods of Sect. 9.6, while the denominator can be derived or computed for the given input distribution. See Exercise 15.

Notice that the central idea is more general: the sensitivity of *any* property of $Y(\mathbf{x})$ for which we can estimate gradients, with respect to *any* property of the input model for which we can compute gradients, along *any* chosen direction $\mathbf{d}$, can be estimated in the same way. See Exercise 14.

Remark 9.8. A quite different situation occurs when the goal is to assess which elements of $\mathbf{x}^\top = (x_1,x_2,\ldots,x_d)$ have a signficant impact on $\mathrm{E}(Y(\mathbf{x}))$ globally as $\mathbf{x}$ varies across a feasible region or region of operability $\mathbf{x} \in \mathsf{C}$. Classical experiment designs, including fractional-factorial and factor-screening designs, were created to sort the elements of $\mathbf{x}$ into significant vs. insignificant effects with statistical confidence. These experiment designs can be substantially more efficient than investigating each element of $\mathbf{x}$ one at a time, as measured by the number of simulation runs required. A good general reference is Tamhane (2009). However, these classical methods from the statistics literature tend to assume data are scarce or expensive, which is not always the case in simulation. Factor-screening designs created specifically for simulation provide more control over the power for detecting significant elements of $\mathbf{x}$ by obtaining simulation outputs sequentially; see, for instance, Sanchez et al. (2009).

9.11 Change of Measure

We typically think of the input distributions that drive the simulation as being tied to the real world, either as it exists or as it will exist after we alter it. Here we discuss the powerful insight that you can simulate under one set of input distributions (probability measures) but reweight the simulation outputs so that they represent performance under a different set of input distributions; this is referred to as a "change of measure." Why would we want to do this? There are at least four contexts of interest (Dong et al. , 2018):

Importance Sampling: A better (lower variance) estimator may be obtained by simulating with other than the target input distributions, often input distributions for which rare but important events are more likely to occur.

Metamodeling: To make predictions over a range of possible input models based only on simulations at a smaller set of distributions.

Changing inputs: Because the real world is changing, the appropriate input models may change, but as they change we would like to leverage some or all of the simulations executed under previous input models.

Target inputs: The real-world inputs are not changing, but our knowledge of their distribution improves as more and more real-world data are revealed, and again we would like to leverage some or all of the simulations executed under previous input models.

The reader may recognize that change of measure appeared in Sect. 9.6.3 for likelihood ratio gradient estimation. For consistency we borrow the notation from that section.

Let $\mathbf{A}(\mathbf{x})$ be an $m \times 1$ input random vector with joint parametric density function $f(\mathbf{a}|\mathbf{x})$, where $\mathbf{x}$ is a vector of distribution parameters.[6] We include the distribution parameters $\mathbf{x}$ as part of the notation $\mathbf{A}(\mathbf{x})$ to make clear the input distribution used to generate it. In Sect. 9.6.3, $\mathbf{A}(\mathbf{x})$ was a vector of exponentially distributed random variables with means $\mathbf{x}^\top = (x_1, x_2, x_3, x_2, x_5)$ representing the $m = 5$ activity times for the SAN. For a fixed set of parameters $\mathbf{x}$, represent the simulation output by $Y(\mathbf{x}) = g(\mathbf{A}(\mathbf{x}))$ for some function g with $\mathbf{A}(\mathbf{x}) \sim f(\mathbf{a}|\mathbf{x})$. Our focus is on the random variable

$$g(\mathbf{A}(\mathbf{x}))\frac{f(\mathbf{A}(\mathbf{x})|\mathbf{x}_0)}{f(\mathbf{A}(\mathbf{x})|\mathbf{x})} = Y(\mathbf{x})W(\mathbf{x}_0|\mathbf{x}), \tag{9.36}$$

where $\mathbf{x}_0$ is also a legitimate set of parameters for f. Throughout this section we refer to $f(\mathbf{a}|\mathbf{x})$ as the *nominal* distribution used for the simulation, and $f(\mathbf{a}|\mathbf{x}_0)$ as the *target* distribution for which we would actually like results. We assume that the support of f does not depend on its parameters, so that if $f(\mathbf{A}|\mathbf{x}) = 0$ then so does $f(\mathbf{A}(\mathbf{x})|\mathbf{x}_0)$. Clearly,

$$\mathrm{E}\left[g(\mathbf{A}(\mathbf{x}))\frac{f(\mathbf{A}(\mathbf{x})|\mathbf{x}_0)}{f(\mathbf{A}(\mathbf{x})|\mathbf{x})}\right] = \int \cdots \int \left[g(\mathbf{a})\frac{f(\mathbf{a}|\mathbf{x}_0)}{f(\mathbf{a}|\mathbf{x})}\right] f(\mathbf{a}|\mathbf{x})\, d\mathbf{a} = \mathrm{E}[Y(\mathbf{x}_0)].$$

That is, even though the inputs $\mathbf{A}(\mathbf{x})$ were generated using input distribution parameters $\mathbf{x}$, the output $Y(\mathbf{x})$ weighted by the likelihood ratio $W(\mathbf{x}_0|\mathbf{x}) = f(\mathbf{A}(\mathbf{x})|\mathbf{x}_0)/f(\mathbf{A}(\mathbf{x})|\mathbf{x})$ has the same expectation as if we had simulated using parameters $\mathbf{x}_0$. Notice that although the *expected values* are the same, the *distributions*

[6] If $\mathbf{A}$ was a discrete-valued random variable, then the same development applies using its mass function.

of the random variables $Y(\mathbf{x}_0)$ and $Y(\mathbf{x})W(\mathbf{x}_0|\mathbf{x})$ are *not* the same, which is the key to importance sampling being able to reduce variance.

Recall that for the SAN the distribution of the time to complete the ith activity is exponential with mean x_i, implying density function $f(a_i|x_i) = \exp\{-a_i/x_i\}/x_i$. If we are interested in the mean time to complete the project, then $g(\mathbf{a}) = \max\{a_1 + a_4, a_1 + a_3 + a_5, a_2 + a_5\}$; if we are interested in the probability that the time to complete the project exceeds t_p, then $g(\mathbf{a}) = I(\max\{a_1+a_4, a_1+a_3+a_5, a_2+a_5\} > t_p)$. In either case, an unbiased estimator for the expected value under parameter $\mathbf{x}_0$ can be obtained by simulating activity times $\mathbf{A}(\mathbf{x})^\top = (A_1, A_2, \ldots, A_5) \sim f(\mathbf{a}|\mathbf{x})$ and recording

$$g(A_1, A_2, \ldots, A_5) \prod_{i=1}^{5} \frac{\exp\{-A_i/x_{0i}\}/x_{0i}}{\exp\{-A_i/x_i\}/x_i}. \tag{9.37}$$

In practice we make n i.i.d. replications $\mathbf{A}_1(\mathbf{x}), \mathbf{A}_2(\mathbf{x}), \ldots, \mathbf{A}_n(\mathbf{x})$, apply (9.37) to each, and average the results.

Notice that by simulating only at a nominal setting $\mathbf{x}$*, we can map out the entire expected-value surface as a function of* $\mathbf{x}_0$. The top two plots in Fig. 9.3 show results from first simulating 10,000 replications of the SAN with activity time means $\mathbf{x}^\top = (1,1,1,1,1)$, and then applying the change of measure to obtain estimates for $\mathbf{x}_0^\top = (1,1,x_{03},1,1)$ with x_{03} ranging from 0.6 to 1.4; estimates of $\mathrm{E}[Y(\mathbf{x}_0)]$ and $\Pr\{Y(\mathbf{x}_0) > 4\}$ are shown along with 95% confidence intervals. The bottom plot also shows estimates of $\Pr\{Y(\mathbf{x}_0) > 4\}$ but with nominal mean $\mathbf{x}^\top = (1,1,0.7,1,1)$ instead. The likelihood ratio when only changing x_{30} simplifies to

$$W(\mathbf{x}_0|\mathbf{x}) = \frac{\exp\{-A_3/x_{03}\}/x_{03}}{\exp\{-A_3/x_3\}/x_3} = \frac{x_3}{x_{03}} \exp\{-A_3(1/x_{03} - 1/x_3)\}.$$

To reiterate, all of the results in the plots were obtained by simulating at a single value of x_3 *and then applying the change of measure.*

Two observations jump out: The widths of the confidence intervals are not uniform across all values of x_{03} in any plot, and the value chosen as the nominal setting affects the precision of the estimates (compare the bottom two plots).

What makes a good nominal distribution to which to apply a change of measure? The answer depends on the purpose of the simulation, in particular whether the goal is to construct a metamodel in $\mathbf{x}_0$ as in the example above, or to achieve a variance reduction at a particular $\mathbf{x}_0$. We focus on finding a good choice to achieve a variance reduction.

Suppose now that we want to estimate $\Pr\{Y(\mathbf{x}_0) > 9\}$ when $\mathbf{x}_0^\top = (1,1,1,1,1)$; project completion times greater than 9 are rare. Based on 1000 replications we obtain a point estimate of 0.0050 ± 0.0044, implying substantial error in the estimate: not even the first significant digit is certain. Simulating instead at $\mathbf{x}^\top = (2,2,2,2,2)$—making the rare event of exceeding 9 much more likely—and then applying the change of measure, we obtain $0.0075 \pm .00049$, roughly an order of magnitude less error. This is the promise of importance sampling.

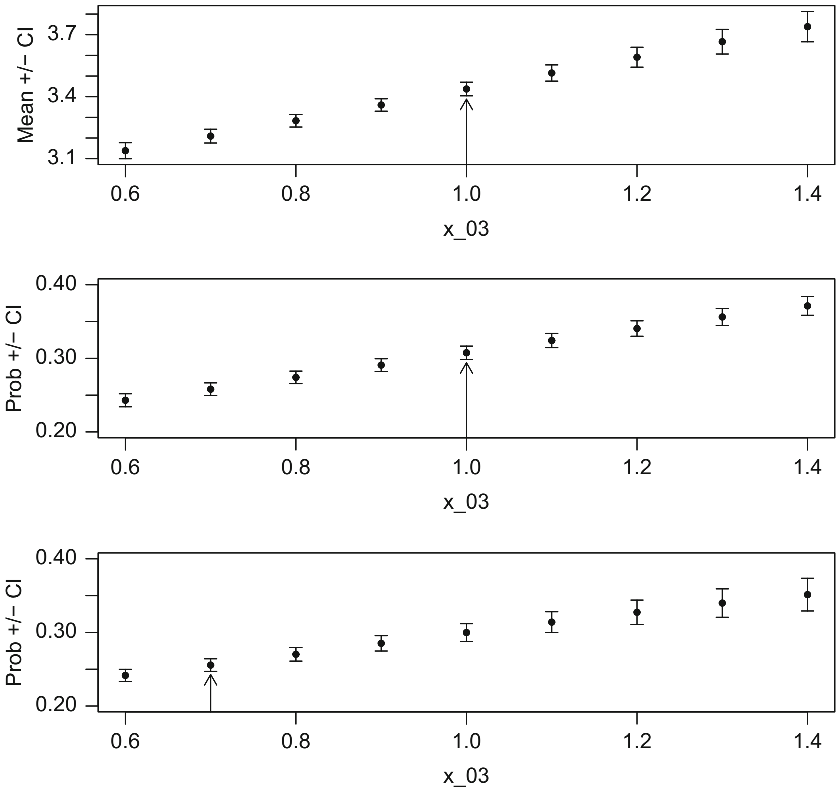

Fig. 9.3 Change-of-measure metamodel for mean time to complete the SAN (top) and probability it takes longer than 4 time units (bottom two) with 95% confidence intervals as a function of the mean of activity 3. The arrow indicates the nominal value of $x_3 = 1$ (top two) or $x_3 = 0.7$ (bottom)

More generally (not just for the SAN example), the change-of-measure estimator based on n i.i.d. replications is

$$\widehat{Y}(\mathbf{x}_0) = \frac{1}{n}\sum_{i=1}^{n} g(\mathbf{A}_i(\mathbf{x}))\frac{f(\mathbf{A}_i(\mathbf{x})|\mathbf{x}_0)}{f(\mathbf{A}_i(\mathbf{x})|\mathbf{x})} = \frac{1}{n}\sum_{i=1}^{n} Y_i(\mathbf{x})W_i(\mathbf{x}_0|\mathbf{x}). \tag{9.38}$$

Let $\mu(\mathbf{x}_0) = \mathrm{E}[Y(\mathbf{x}_0)]$. Then by direct application of the defintion of variance (see Exercise 19)

$$\mathrm{Var}[\widehat{Y}(\mathbf{x}_0)] = \frac{1}{n}\left[\int\cdots\int \frac{(g(\mathbf{a})f(\mathbf{a}|\mathbf{x}_0))^2}{f(\mathbf{a}|\mathbf{x})}\,d\mathbf{a} - \mu(\mathbf{x}_0)^2\right]. \tag{9.39}$$

To estimate (9.39) we use the sample variance in the usual way, since (9.38) is the average of i.i.d. outputs $Y_i(\mathbf{x})W_i(\mathbf{x}_0|\mathbf{x}), i = 1,2,\ldots,n$.

Apparently the variance (9.39) will be small if the ratio $(g(\mathbf{a})f(\mathbf{a}|\mathbf{x}_0))^2/f(\mathbf{a}|\mathbf{x})$ tends to be small across the range of $\mathbf{A}$. In particular, in regions where $(g(\mathbf{a})f(\mathbf{a}|\mathbf{x}_0))^2$

is large, we want the nominal distribution $f(\mathbf{a}|\mathbf{x})$ also to be large; this is what we mean by choosing a nominal distribution that assigns more probability to "important" regions. In fact, one can show that the variance-minimizing choice of nominal distribution $f(\mathbf{a}|\mathbf{x})$ is proportional to $|g(\mathbf{a})|f(\mathbf{a}|\mathbf{x}_0)$. Unfortunately, knowing this is not as helpful as it might seem, because g is typically not a simple function as in the SAN example, but rather is implied by a complex computer code.

The SAN is also a nice problem for change of measure for another reason: each replication generates exactly five independent inputs (activity times), so the likelihood ratio for each replication is the product of exactly five terms. Consider instead the waiting times of the first m customers in an $M/G/1$ queue:

$$Y_i(\mathbf{x}) = \max\{0, Y_{i-1}(\mathbf{x}) + S_{i-1}(\mathbf{x}) - A_i(\mathbf{x})\},\ i = 1,2,\ldots,m,$$

where S_i is the service time of the ith customer, A_i is the interarrival time between customers $i-1$ and i, $\mathbf{x}$ are the parameters of the service-time and interarrival-time distributions with densities $f_S(s|\mathbf{x})$ and $f_A(a|\mathbf{x})$, respectively, and $Y_0 = X_0 = 0$. We know f_A is an exponential distribution, and suppose that f_S is Weibull, so $\mathbf{x}^\top = (\lambda, \alpha, \beta)$. We compute the sample mean, $\overline{Y}(\mathbf{x}) = \sum_{i=1}^m Y_i(\mathbf{x})/m$, but would like to apply a change of measure to obtain an estimate under a different set of parameters $\mathbf{x}_0$. The likelihood ratio by which we multiply $\overline{Y}(\mathbf{x})$ is

$$W(\mathbf{x}_0|\mathbf{x}) = \frac{f_A(A_1|\mathbf{x}_0)}{f_A(A_1|\mathbf{x})} \prod_{i=2}^{m} \frac{f_S(S_{i-1}|\mathbf{x}_0) f_A(A_i|\mathbf{x}_0)}{f_S(S_{i-1}|\mathbf{x}) f_A(A_i|\mathbf{x})}. \tag{9.40}$$

It can be shown (e.g., Owen (2019)) that the variance of $W(\mathbf{x}_0|\mathbf{x})$ will tend to increase exponentially in m, which may make the change of measure useless if the number of simulated customers m is large, a common situation in queueing simulation. This illustrates that change-of-measure strategies can fail, and good ones are often problem dependent. For instance, since the $M/G/1$ queue is regenerative (see the Appendix in Chap. 5), and regenerative cycles are independent, the change of measure can be computed and applied individually to each (relatively short) cycle rather than globally to the overall sample mean as in (9.40). See for instance Goyal et al. (1987).

Importance sampling is a powerful variance reduction technique that can make some rare-event estimation problems possible that would be impossible otherwise; on the other hand it can go horribly wrong and inflate variance (see Fig. 9.3). Fortunately there is a vast literature on effective importance sampling for many classes of problems. For excellent general references and pointers into the literature see Owen (2019) and Asmussen and Glynn (2007). Dong et al. (2018) discuss the other three reasons for using change of measure, metamodeling, changing inputs, and target input.

Remark 9.9. That $f(\mathbf{a}|\mathbf{x})$ and $f(\mathbf{a}|\mathbf{x}_0)$ have the same suppport is a sufficient condition to use change of measure, but not necessary. The weaker condition that $g(\mathbf{a})f(\mathbf{a}|\mathbf{x}_0) \neq 0$ whenever $f(\mathbf{a}|\mathbf{x}_0) > 0$ is adequate.

Exercises

1. Recall the paired-t confidence interval (9.8) for the difference $\theta(\mathbf{x}_1) - \theta(\mathbf{x}_2)$. Show that
$$S_D^2 = S^2(\mathbf{x}_1) + S^2(\mathbf{x}_2) - 2S(\mathbf{x}_1, \mathbf{x}_2),$$
where $S^2(\mathbf{x}_1)$ and $S^2(\mathbf{x}_2)$ are the marginal sample variances, and
$$S(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{n-1} \sum_{j=1}^{n} \left(Y_j(\mathbf{x}_1) - \overline{Y}(\mathbf{x}_1)\right) \left(Y_j(\mathbf{x}_2) - \overline{Y}(\mathbf{x}_2)\right)$$
is the sample covariance. This shows how the paired-t interval captures the effect of common random numbers.
2. Consider the following five scenarios for the mean activity times in the SAN example. Make 100 replications of each one and apply subset selection with 95% confidence to see which if any can be eliminated from being the best in terms of the smallest expected value of the time to complete the project. Now repeat with 200 replications of each.

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\mathbf{x}_5$
x_{i1}	0.5	1	1	0.3	1
x_{i2}	1	0.5	1	1	1
x_{i3}	1	1	0.5	1	1
x_{i4}	1	1	1	0.4	1
x_{i5}	1	1	1	1	0.5

3. For the same scenarios as in Exercise 2, apply the selection-of-the-best procedure to find the scenario with the smallest expected completion time with 95% confidence. Use $n_0 = 10$ and $\delta = 0.1$. Run the experiment with and without using common random numbers and comment on the number of replications required to reach a decision in each case.
4. Implement all of the gradient estimators described in this chapter for the SAN example. Using $\tau_j = c_j = 1$ and $\ell_j = 0.5$ for $j = 1, 2, 3, 4, 5$, and $b = 1$, select several feasible settings for $\mathbf{x}$ and estimate the gradient using each method. Compare the methods in terms of the variability of the gradient estimates. Use $\Delta x = 0.1$ in FD.
5. When applying stochastic approximation to the SAN optimization, suppose that at iteration i we find $\sum_{j=1}^{5} c_j(\tau_j - x_{i+1,j}) > b$. First derive $\mathbf{x}'_{i+1}$, the perpendicular projection of $\mathbf{x}_{i+1}$ onto the plane $\sum_{j=1}^{5} c_j(\tau_j - x_j) = b$. Then provide a refinement that insures that $\mathbf{x}'_{i+1}$ also satisfies the lower bound constraints $x'_{i+1,j} \geq \ell_j$, $j = 1, 2, 3, 4, 5$; this need not be a point that is on the plane, but it should be feasible.
6. Using each of the gradient estimators described in this chapter, implement a stochastic approximation search for the optimal mean activity times for the SAN example using the parameters in Exercise 4. Suggested settings for this

problem are $a_i = 1/i$, starting scenario $\mathbf{x}^\top = (1,1,1,1,1)$, and at least 50 replications per gradient estimate, but you should experiment with these values.

7. Using linear programming software, solve the SAA problem (9.28) using $\tau_j = c_j = 1$ and $\ell_j = 0.5$ for $j = 1,2,3,4,5$, and $b = 1$. Use $n = 10$ replications if you have to set it up manually, and $n = 10, 100, 1{,}000$ if you can automate the set- up.
8. For the setting described in Sect. 9.3.4 when there is a unique best scenario, prove that any allocation with all $\beta_i > 0$ will drive the probability of incorrect selection to 0 as $N \to \infty$. Hint: When there is a unique best, then there is an $\varepsilon > 0$ such that $\theta_i - \theta_B \geq \varepsilon$ for all inferior scenarios i. Now apply the strong law of large numbers to show that for large enough N the best will be always be identified.
9. For the special case of $\theta_B = \theta - \delta$, $\theta_i = \theta$ for all $i \neq B$, and $\sigma_i^2 = \sigma^2$ for all i, derive (9.13) and (9.14) from (9.11) and (9.12).
10. Recall Remark 9.2. For the special case of $K = 2$ show that the rate-optimal allocation satisfies $(\beta_B/\sigma_B)^2 = (\beta_i/\sigma_i)^2$.
11. If $Z \sim \mathrm{N}(0,1)$, then it can be shown that $\mathrm{E}[\max\{0, \mu + \sigma Z\}] = \mu\Phi(\mu/\sigma) + \sigma\phi(\mu/\sigma)$, where Φ and ϕ are the standard normal cdf and density, respectively. In the Bayesian ranking-and-selection setting of Chen and Ryzhov (2019) the posterior distribution of $(\Theta(\mathbf{x}_i), \Theta(\mathbf{x}_j))$ given the history $\mathscr{H}$ is normal with means $\overline{Y}(\mathbf{x}_i), \overline{Y}(\mathbf{x}_j)$, variances $\sigma^2(\mathbf{x}_i)/n(\mathbf{x}_i), \sigma^2(\mathbf{x}_j)/n(\mathbf{x}_j)$, and covariance zero. Use this fact to derive an expression for EI.
12. Many programming languages, including Python, have the capability to parallelize loops on any computer with multiple cores/threads, provided the computations within each iteration of the loop do not interact. The subset selection procedure is a good candidate for such parallelization, since the simulation jobs $\mathsf{Job}_1, \mathsf{Job}_k 2, \ldots, \mathsf{Job}_K$ in Sect. 9.9 do not interact. With a little care the same is true for select the best. Using whatever language you have available, parallelize the simulation loops for these two ranking-and-selection algorithms, and evaluate the speed-up as K increases using the language's tools for timing computations.
13. A convenient test problem for simulation optimization is the following: Consider an $M/M/1$ queue with arrival rate λ and service *rate* (not mean) x. Suppose that it costs \$$c_1$ per unit of service rate, and \$$c_2$ per unit of steady-state mean time spent in the system (that is, it costs more to have faster service but it also costs more to deliver slow service). Using results for the $M/M/1$ queue, the mean total cost is $c_1 x + c_2/(x - \lambda)$. Thus, test problems for minimizing the mean cost using simulation optimization can be created with known solutions. Reasonable choices are $\lambda = 1$, $1.2 \leq x \leq 21$, $c_1 = 1$, and $c_2 = 36$, where x can be treated as continuous or discrete. Use this problem to practice some of the simulation optimization methods in this chapter.
14. The sensitivity measure (9.35) might be called "mean sensitivity to mean." Extend the definition to "mean sensitivity to variance;" that is, the sensitivity of the expected value of the output to the *variance* of the input. Assume that the

input has a gamma distribution and we will use the steepest ascent direction for the variance of the input.

15. For the SAN example, suppose that activity 3 has a gamma distribution with parameters $(\alpha_0, \beta_0) = (2, 2)$, while all of the other activities are exponentially distributed with mean 1. Using one of the gradient estimation methods in Sect. 9.6 estimate the sensitivity of the mean time to complete the project to the mean and variance of activity 3 along the steepest ascent direction in each case. You will need the result from Exercise 14.
16. For the SAN example in Sect. 9.11, run intensive simulations at each x_3 value in Fig. 9.3 to estimate the true values of $\mathrm{E}[Y(\mathbf{x}_0)]$ and $\Pr\{Y(\mathbf{x}_0) > 4\}$. Compare these to the results from simulating only at nominal $\mathbf{x}^\top = (1,1,1,1,1)$ using 10,000 replications and then applying the change of measure.
17. Following on from Exercise 16, try the nine settings of $\mathbf{x}^\top = (1,1,x_3,1,1)$ to find one that minimizes the maximum of the nine confidence interval halfwidths.
18. For the SAN example in Sect. 9.11, run intensive simulations at each x_3 value in Fig. 9.3 to estimate the true values of $\mathrm{E}[Y(\mathbf{x}_0)]$ and $\Pr\{Y(\mathbf{x}_0) > 4\}$. Compare these to the results from simulating 5000 replications at nominal $\mathbf{x}^\top = (1,1,0.7,1,1)$ and 5000 at $\mathbf{x}^\top = (1,1,1.2,1,1)$ and then averaging the results after applying the change of measure. See Dong et al. (2018) for other ways to use change of measure to combine results from more than one nominal simulation.
19. Derive Eq. (9.39). Suppose that $g(\mathbf{a}) > 0$ for all $\mathbf{a}$. Show that the nominal density $f^\star(\mathbf{a}) = g(\mathbf{a}) f(\mathbf{a}|\mathbf{x}_0)/\mu(\mathbf{x}_0)$ leads to a zero-variance importance-sampling estimator. Going further, show that the variance-minimizing distribution $f^\star$ is proportional to $|g(\mathbf{a})| f(\mathbf{a}|\mathbf{x}_0)$ in general.

Chapter 10
Simulation for Research

This book is about simulation modeling, programming, and experimentation for the purpose of systems analysis. However, stochastic simulation is also a tool that can be used to support basic research in domains such as simulation, optimization, queueing, financial engineering, statistical learning, healthcare, production planning and logistics. In this chapter we cite some papers that demonstrate effective application of simulation in research and use them to highlight general principles and practices. We start with two important distinctions.

The first distinction is between a *practitioner's experiment* and a *researcher's experiment*. This book is written primarily from the point of view of the practitioner's experiment. The practitioner has a real problem to solve and has some limit on how much time or effort they can expend to solve it. The practitioner will try to build an appropriate simulation model or models, will perform the simulation experiment using the best methods they know or have access to, and will use the results to solve their problem. If they learned simulation from this (or many other) books then they will also try to assess the quality of their answer (e.g., via a confidence interval) or they will use a procedure that is designed to deliver a certain level of quality (e.g., a ranking-and-selection procedure with a guaranteed probability of correct selection). They will, however, only have a vague sense as to whether they got the "right answer" to their problem based on what happens when they apply their simulation results in the real world.

The researcher's experiment, on the other hand, is driven by an abstract research question whose answer may be useful in practice, but is not the answer to a specific practical problem. Because it is not directly tied to the real world, which is messy, a research question can be formulated and answered very precisely. The researcher's experiment may, and often will, involve solving the same problem repeatedly for simulated instances that differ through some controllable experimental factors and in the random numbers used in the simulation. For instance, Sect. 10.1 below describes generating random optimization problems of a particular type and solving each instance with the same algorithm to assess the algorithm's performance over a space of problems.

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*,
International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0_10

Unlike the practitioner's experiment, the researcher's experiment frequently involves simulating scenarios for which the answer is known in advance; knowing the answer facilitates a better evaluation. For example, Sect. 10.3 describes an experimental study of new ranking-and-selection procedures. Ranking-and-selection procedures are supposed to discover the best scenario with a guaranteed probability of correct selection. The achieved probability of correct section for a new procedure can be estimated by applying it repeatedly (with different random numbers) to simulated scenarios for which the true best scenario is already known. This contrasts with the practitioner who will apply ranking and selection once on a collection of scenarios for which the best is not known and then actually implement their selection.

A second distinction is between simulation *experiments* that support research as opposed to *illustrations* or examples. What is the difference? A successful experiment allows the researcher to make statements about cases, scenarios, or problems that they *did not* try based on cases, scenarios, or problems that they *did* try. To accomplish this an experiment must be *designed* so as to provide inference about a well-defined space of cases, scenarios, or problems. An illustration, on the other hand, is a specific case, scenario, or problem that helps the reader understand how a method is implemented and how the results should be interpreted.

This book is full of illustrations, starting with those in Chap. 3 that were used to demonstrate simulation methods throughout the book. These illustrations were (we hope) useful for learning and understanding the methods, but by themselves they do not prove that the methods work more generally.

In the following sections we provide some guidelines for generating random test problems, conducting robustness studies, linking experimental factors and measuring and controlling error in a research simulation experiment. We also comment on the need for reproducible results. In addition to the research papers cited, this chapter also draws on Sect. 3 of Goldsman et al. (2002) which discusses evaluation of simulation output analysis procedures.

10.1 Random Test Problems

Simulation is often used to generate random test problems to support saying something general about the performance of a solution method. Consider the following example:

The two-dimensional knapsack problem is a deterministic (and difficult) combinatorial optimization problem of the form

$$\max \sum_{j=1}^{n} c_j x_j \tag{10.1}$$

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i,\ i = 1, 2$$

$$x_j \in \{0, 1\},\ j = 1, 2, \ldots, n,$$

where $c_i > 0$ and $a_{ij} \geq 0$. Think of c_j as the value of selecting the jth item ($x_j = 1$), while a_{1j} and a_{2j} are two types of "costs" for selecting it (e.g., weight and price), and the totals of these "costs" are constrained by b_1 and b_2, respectively.

Hill and Reilly (2000) compared the performance (speed and quality of solution) of a heuristic and a convergent algorithm for solving (10.1) by applying them to a large number of randomly generated test problems. A test problem is specified by providing values for c_j, a_{ij}, b_j, and n. Hill and Reilly (2000) fixed the number of items at $n = 100$ and set b_i to be a specified fraction of $\sum_{j=1}^{n} a_{ij}$. This left them with the need to decide how to randomly generate values of c_i and a_{ij} to draw meaningful comparisons.

A prescription that seems reasonable is to select uniform distributions for c_j, a_{1j} and a_{2j}, and then to randomly and independently generate the necessary values. We call these "completely random test problems." Assuming that we are comfortable setting the ranges on these uniform distributions, then the space specified by the ranges will be randomly covered. But is that enough?

Hill and Reilly (2000) showed conclusively that the answer is "no" by demonstrating that *correlation* across the coefficients (c_j, a_{1j}, a_{2j}) can have a dramatic effect on optimization performance, and these correlations have a practical meaning. For instance, a strong positive correlation between a_{ij} and c_j means that if the jth item is valuable, then it also tends to be costly; and this makes the problem hard because desirable items tend to break the budget. Similarly, strong negative correlation between a_{1j} and a_{2j} means that an item that is inexpensive relative to the budget b_1 tends to be expensive relative to budget b_2; this also makes the problem difficult. Further, the correlation structure affects the heuristic and convergent algorithm differently. While some problems that exhibit these features will be generated, by chance, for completely random test problems, they will be rare and their effect will be averaged over all of the (mostly) low-correlation test problems. Therefore, this important, and practically justifiable, feature will not be identified.

Hill and Reilly (2000) illustrates our first principle of simulation for research:

Research principle 1. *Completely random cases, scenarios, or problems are not necessarily relevant ones. Instead, generate test cases, scenarios, or problems that have features that are representative of the space of interest.*

Hill and Reilly (2000) carefully controlled the strength of the correlations in their random test problems via a designed experiment. This allowed them to make much stronger statements about how the solutions methods will behave in practice. Variate-generation methods such as NORTA (Sect. 6.3.2) can be used to induce such correlation.

This example also illustrates an obvious, but important second principle:

Research principle 2. *If you are comparing things, use common random numbers.*

Hill and Reilly (2000) could have generated the test problems to evaluate the heuristic independently from those used to evaluate the convergent algorithm. That is, they could have chosen the correlation settings at which they wanted to do experiments, but used different random numbers to actually generate the coefficients for the problems solved by the heuristic and for the problems solved by the convergent algorithm. The comparisons would still have been with respect to the same problem space, but not with respect to precisely the same knapsack problems. If they had used independent test sets, then some of the observed differences in performance would have been due to solving different test problems; this source of variation is eliminated by using common random numbers since the problem sets will be identical.

10.2 Robustness Studies

Simulation experiments are also used to assess the robustness of a research result. By "robustness" we mean the ability of the procedure, method, or concept to perform as expected across a wide range of conditions.

Controlled experimentation is a topic addressed by the statistical design of experiments; see, for instance Montgomery (2009). Experiment design starts by identifying the factors that might affect the performance of a product, process, or treatment and then varies these factors in a systematic fashion. Careful identification of the appropriate factors, both favorable and unfavorable, is critical to a study of robustness, as illustrated in the following example.

Iravani et al. (2005) proposed indices for measuring the structural flexibility of a manufacturing or service operation. Flexibility in their context is the capability of the operation to satisfy multiple types of demand with available resources in the face of changing demand and resource capacity. The "structural" in structural flexibility refers to the choice of how many and which types of demand that each resource can satisfy. In a manufacturing setting where there are N factories and K types of products (e.g., car models), structural flexibility depends on which of these products each factory can produce. In a service setting like a call center where there are N agents and K types of calls, structural flexibility depends on which types of calls each agent is cross-trained to handle. Iravani et al. (2005) proposed an easy-to-compute structural flexibility index for strategic-level comparison of competing system designs, an index that should predict which system design would be most productive in a changing and unpredictable real-world environment.

In brief, Iravani et al. (2005) represented a system design as a graph from resources $\{S_1, S_2, \ldots, S_K\}$ to demand types $\{D_1, D_2, \ldots, D_N\}$ with an arc from $S_k \rightarrow D_j$ if resource k can satisfy some demand of type j. Their two structural flexibility (SF) indices are functions of the number of nonoverlapping paths through this network by which excess capacity to satisfy demand type i can be redirected to satisfy demand of type j, and also the number of different resources that can satisfy demand of type j, for $i, j = 1, 2, \ldots, K$. The research question is, how robust is this simple measure for ranking the performance of system designs relative to the types of real-world systems to which it might be applied?

To answer this question Iravani et al. (2005) employed a designed simulation experiment. The key to designing an experiment that convincingly establishes robustness is that it varies factors that are *not* inputs to the SF measure, but do represent what occurs in the real world. These included the following:

Physical flow in the system: An open, parallel queueing environment (such as a call center) and a closed serial queueing environment (such as a production line operating with a constant work in process or WIP) were chosen.

How close the system is to capacity: Congestion levels were controlled in the open queueing system by setting the demand arrival rate to achieve specific levels of utilization, and in the closed system by setting the WIP to achieve different levels of jobs/worker.

Uncertainty in the environment: Short-term fluctuations and long-term changes were considered:

- **Short-term variation**: Interarrival times and processing times were modeled as having gamma distributions, with different levels of variability attained by setting their coefficient of variation (standard deviation divided by the mean).
- **Long-term variation**: Shocks randomly inject a persistent change in demand. Shocks that increase one type of demand at the expense of another, and shocks that simply increase one type of demand were included.

The simulation experiment was conducted on pairs of system designs to see if the one with the higher SF index (which is computable without simulation) was also the one with the higher productivity (as estimated via simulation).

This paper illustrates another research principle:

Research principle 3. *Run a designed experiment that identifies and controls factors that might influence the outcome, both favorably and unfavorably, and that are actually encountered in practice.*

10.3 Linking Experimental Factors

In physical experiments the operable range for each experimental factor, such as temperature, pressure, or dosage, can often be set based on the nature of the system. Unfortunately, this may not be the case when evaluating, say, a new simulation output analysis method that is supposed to apply to any problem in a large and diverse class. Therefore, rather than independently specifying a range for each factor, it may be better to link the ranges of the factors in meaningful ways. Here we provide one illustration.

Ranking-and-selection procedures are simulation optimization methods whose objective is to discover the best simulated system, in terms of largest or smallest true expected performance, with some guaranteed probability of correct selection. See Sect. 9.3 for background on ranking and selection. Nelson et al. (2001) proposed and then empirically evaluated ranking-and-selection procedures created for simulation optimization problems where the number of alternative scenarios K is moderate (up to 500). Their interest was primarily in computational efficiency of the procedures as measured by the number of simulation replications required to deliver the guarantee.

A number of factors in addition to K can affect the efficiency of a ranking-and-selection procedure; Nelson et al. (2001) focused on four: The values of the true means, $\mu_1, \mu_2, \ldots, \mu_K$, the output variances of each scenario, $\sigma_1^2, \sigma_2^2, \ldots, \sigma_K^2$, the initial ("first stage") number of replications obtained from each scenario n_0, and the size of the indifference-zone parameter δ. Recall that ranking and selection procedures are often sequential and therefore have multiple stages, and that δ is the smallest practically significant difference in mean performance that the user is interested in detecting. In their experiments, Nelson et al. (2001) linked these factors to allow general conclusions to be drawn.

Notice that a ranking-and-selection problem will be computationally easy if the mean values are widely separated, and if the variances of the outputs are small while δ is large relative to the differences in the means. On the other hand, if the means are close, the variances large, and the difference we care about is small, then a great deal of simulation effort will be required to guarantee finding the best. These insights are useful for linking the factors.

Nelson et al. (2001) let $\mu_1 = \delta$ be the largest mean (bigger was better in their paper) and anchored the experiment design to this value. They considered a number of configurations of the means, including:

Slippage configuration: $\mu_1 = \delta$, or a multiple of δ, and $\mu_2 = \mu_3 = \cdots = \mu_K = 0$. This is a difficult configuration since all inferior scenarios are tied for second best.

Monotone decreasing means: $\mu_1 = \delta$, and $\mu_i = \mu_1 - (i-1)\delta/\tau$, $i = 2, 3, \ldots, K$, with $\tau = 1, 2, 3$. These are easier configurations since many scenarios are substantially inferior to the best.

They then set $\delta = d\sigma_1/\sqrt{n_0}$, a multiple of the standard deviation of the first-stage sample mean of the best system, varying $d = 1/2, 1, 2$. This ties the separation between the best and the rest to the precision with which we can estimate the mean of the best.

Finally, for the variances of the systems, they considered situations in which the variance of the best system is higher or lower than the inferior systems, as follows:

Common variance configuration: $\sigma_2^2 = \sigma_3^2 = \cdots = \sigma_K^2 = \sigma^2$ and $\sigma_1^2 = \rho\sigma^2$, with $\rho = 1/2, 2$.

Unequal variance configuration: Variances increase as the mean decreases, $\sigma_i^2 = |\mu_i - \delta| + 1$, and variances decrease as the mean decreases, $\sigma_i^2 = 1/(|\mu_i - \delta| + 1)$, $i = 1, 2, \ldots, K$.

Notice that the number of factors has now been effectively reduced to three: The number of scenarios K and first-stage sample size n_0, both factors for which a reasonable range of operability can be determined, and σ_1^2, the variance of the output from the best system, which can be set to any fixed value (say 1) because all of the other factors are relative to it. Thus, conclusions from the empirical study are with respect to the *relative relationships* among the means, variance, and indifference-zone of a simulation optimization problem, not their actual values.

Research principle 4. *When there are no natural ranges of operability for your factors, link them so that they are large or small relative to each other.*

Nelson et al. (2001) generated simulation output data directly from normal and lognormal distributions; this made it easy to control the experimental factors described above, and to know which scenario is actually the best. Controlling experimental factors is often easier with surrogate models than with realistic discrete-event simulations; surrogate models have characteristics like realistic simulations but are simple, tractable, and controllable. In this book we have often used the AR(1) as a surrogate for output from a steady-state simulation.

While well-chosen surrogate models share characteristics with real simulations, we cannot be certain that they include all characteristics that appear in, say, a complex supply chain simulation, and that might matter. For this reason, running experiments with models that are less controllable, but a step closer to realism, is often appropriate. Tractable queueing models (e.g., $M/M/\infty$, $M/G/1$), simple inventory models (e.g., (s,S) inventory model with Poisson demand) and simple financial options (e.g., European call or put) are common choices. A great source of test problems for simulation optimization is the SimOpt library https://github.com/simopt-admin/simopt/wiki.

Research principle 5. *Since it may not be possible to anticipate all important factors, include some realistic examples along with the controllable surrogate models.*

10.4 Controlling Error in Research Simulations

Recall the Pollaczek–Khinchine formula (Eq. (3.4)) which gives the steady-state expected waiting time in an $M/G/1$ queue. A related formula provides the steady-state expected number of customers in the queue:

$$q = \frac{\lambda^2(\sigma^2+\tau^2)}{2(1-\lambda\tau)} = \frac{\rho^2(c^2+1)}{2(1-\rho)},$$

where λ is the arrival rate of the Poisson arrival process and (τ,σ^2) are the mean and variance of the service-time distribution; or equivalently $\rho = \lambda\tau$ is the traffic intensity (a measure of system load), and $c = \sigma/\tau$ is the coefficient of variation of the service time. Typically we use this formula to predict the expected number in the queue given estimates for λ, τ, and σ^2. However, it is clear that if instead we had estimates of the service-time parameters (τ,σ^2) and the average number of customers in the queue q then we could use this formula to solve for what the arrival rate λ must have been. Having this estimated arrival rate we could then predict what the expected number in the queue would be if we changed the service process (e.g., lowered the mean service time, τ).

Whitt (1981) considered a more general version of this situation: Suppose that the congestion in a real-world queueing system can be observed, as well as the queue's service process, but the arrival process—which may be a very general stationary arrival process—is not observed or easily characterized. We want to know what the expected number in the system would be if the same arrival process encountered an altered service process, new equipment or people, for instance.

Whitt's goal was to use the available information to construct a robust, simple, and tractable renewal-process approximation of the unknown arrival process that could then be the input to a queueing model. "Robust" in this context means that the queueing model will provide an accurate approximation of the steady-state congestion under the new service process; "simple" means that we only need two moments (arrival rate and the coefficient of variation of the time between arrivals) to fit the approximating renewal process; while "tractable" means that using this approximating renewal process we can compute the steady-state expected number in the system using something like a Pollaczek–Khinchine formula.

For instance, suppose that the observable queueing system is a $G/M/1$: general stationary arrival process G (which may include dependence), exponential service-time distribution and one server. The service rate and some measures of congestion for this system are observable. The altered system will be a $G/M'/1$, where M' is an exponentially distributed service process with a different service rate. The new system will be approximated by a $GI/M'/1$ queue; GI denotes "general, independent" which means a renewal arrival process. The objective is to have GI approximate G in such a way that the $GI/M'/1$ gives accurate congestion predictions for the $G/M'/1$. *A key point of the paper is that the quality of the arrival-process approximation is judged by how accurately the queueing model predicts steady-state congestion.*

To provide a fair evaluation, Whitt (1981) considered queues with arrival processes that were not of the same type as his approximating renewal processes. For instance, the Poisson arrival process was one of his approximations. Therefore, the $M/M/1$ queue was not an interesting test case because the true arrival process and the approximating arrival process are of the same type (Poisson). Instead, his test cases had arrival processes for which the steady-state expected number in system is not known.

How do you evaluate the accuracy of an approximation when you do not know the true value that it is approximating? Whitt (1981) used simulation to estimate the true expected number in the queue for his test cases; simulation is another form of approximation. To be valid, the error in the simulation estimate has to be small enough that it can be considered insignificant. A feature of simulation is that we can quantify the error in our estimates, and we can drive this error to 0 by increasing the run length or number of replications. Section 8.1 described methods for controlling the error in this way.

Research principle 6. *If you reduce the standard error or confidence interval width enough, then the simulation estimate is effectively the same as the true value.*

To be more concrete, suppose that Whitt's approximation is considered good enough if it has no more than 10 % error relative to the true value:

$$\frac{|q_{\text{approx}} - q_{\text{true}}|}{q_{\text{true}}} \overset{?}{\leq} 0.1.$$

Instead of the true value, we have $\widehat{q}_{\text{true}}$, a simulation estimate of it; $\widehat{q}_{\text{true}}$ has estimated error of, say, $\pm H$, which depends on the run length or number of replications. This error must be made small enough that we can conclude that the approximation is, or is not, within 10 % relative error even considering the estimation error. Stated differently, the ratio above should be satisfied, or not satisfied, for every possible value of q_{true} such that $\widehat{q}_{\text{true}} - H \le q_{\text{true}} \le \widehat{q}_{\text{true}} + H$. In Whitt (1981) this meant that different run lengths were required depending on the traffic intensity in the simulated queue.

We next consider a different aspect of controlling error, and a different paper. Sections 10.1–10.3 described choosing and controlling the important factors in a research simulation. This is not enough, however, when empirical evaluation of performance is estimated through repeated trials, as is common in the study of new statistical procedures. The illustration we will use is the evaluation of control-variate estimators, which were described in Sect. 8.3.

In brief, a control-variate estimator $\widehat{\beta}_0$ is an alternative to the sample mean $\bar{Y}$ for estimating the expected value $\mu_Y = \mathrm{E}(Y)$. As the number of replications goes to infinity, control-variate estimators are asymptotically consistent for μ_Y and have smaller variance than $\bar{Y}$; but in finite samples they may be biased and could have larger variance. Suppose we wanted to study how control-variate estimators behave when the sample size is small. We might be interested in properties such as bias, $\mathrm{E}(\widehat{\beta}_0 - \mu_Y)$, and variance, $\mathrm{Var}(\widehat{\beta}_0)$. Since there is an associated confidence interval for μ_Y, $\widehat{\beta}_0 \pm t_{1-\alpha/2,n-2}\sqrt{\widehat{\Sigma}_{11}}$, we might also be interested in its coverage probability and how well $\widehat{\Sigma}_{11}$ works as an estimator of $\mathrm{Var}(\widehat{\beta}_0)$. Experiments such as the ones described below were used to study control-variate estimators in Nelson (1990).

Listing the factors that might affect the performance of control-variate estimators would provide a number of instances to simulate (see Nelson (1990)). Most likely we would select instances for which μ_Y is known. For a particular instance, simulating it one time yields one set of estimates $\widehat{\beta}_0^{(1)}$, $\widehat{\Sigma}_{11}^{(1)}$, and $\widehat{\beta}_0^{(1)} \pm t_{1-\alpha/2,n-2}\sqrt{\widehat{\Sigma}_{11}^{(1)}}$, where we use a superscript (i) to indicate the ith trial. To estimate bias and variance, we need multiple trials or "macroreplications" of the same instance, $\widehat{\beta}_0^{(1)}, \widehat{\beta}_0^{(2)}, \ldots, \widehat{\beta}_0^{(m)}$. These will be i.i.d. because they are simulations of the same problem instance, but with different random numbers on each macroreplication. The bias, variance, and coverage of the control-variate estimator for this instance can be estimated, respectively, by

$$\widehat{b} = \frac{1}{m}\sum_{i=1}^{m}\widehat{\beta}_0^{(i)} - \mu_Y = \bar{\beta}_0 - \mu_Y$$

$$S^2_{\widehat{\beta}_0} = \frac{1}{m-1}\sum_{i=1}^{m}\left(\widehat{\beta}_0^{(i)} - \bar{\beta}_0\right)^2$$

$$\widehat{p} = \frac{1}{m}\sum_{i=1}^{m} I\left\{\mu_Y \in \widehat{\beta}_0^{(i)} \pm t_{1-\alpha/2,n-2}\sqrt{\widehat{\Sigma}_{11}^{(i)}}\right\}.$$

Each of these is an estimate, and therefore subject to error, but they are also averages of i.i.d. observations so we know how to measure and control the error to be small enough to support the conclusions we want to reach; m should not be chosen arbitrarily.

For instance, suppose that we set $1-\alpha = 0.95$, meaning 95 % confidence intervals. Even if the confidence interval has the desired coverage, the standard error of $\widehat{p}$ is $\sqrt{(0.95)(0.05)/m}$. If $m = 30$ then two times this standard error is $\approx$0.08, which means we cannot really distinguish coverage of 0.95 from 0.87. To get coverage estimates with approximately two decimal places of precision we need m to satisfy

$$2\sqrt{\frac{(0.95)(0.05)}{m}} < 0.01$$

or $m \approx 1900$.

What about assessing the quality of $\widehat{\Sigma}_{11}$? Similar to the situation in Whitt (1981), the true $\mathrm{Var}(\widehat{\beta}_0)$ is not known. However, $S^2_{\widehat{\beta}_0}$ is an unbiased estimator of it, since it is a direct estimator based on i.i.d. observations of $\widehat{\beta}_0$. Therefore, an estimator of the bias of $\widehat{\Sigma}_{11}$ is

$$\frac{1}{m}\sum_{i=1}^{m}\widehat{\Sigma}_{11}^{(i)} - S^2_{\widehat{\beta}_0}.$$

The analysis described above is embedded within the larger experiment that varies the factors that affect control-variate estimators; that is, it is an experiment nested inside an experiment. While the practitioner faces one problem instance, and solves it $m = 1$ times, the researcher creates (systematically or randomly) many instances, and simulates each to an acceptable level of error to draw valid conclusions. This gives the research principle

Research principle 7. *Measure and control the error in your research experiment using nested simulations.*

10.5 Reproducibility of Results

Research papers include research results, and our trust in those results is bolstered by external evaluation by experts before acceptance for publication, the so-called peer review process. For instance, when the result is a new theorem, then a proof of the theorem is included so that it can be verified (or reproduced) by the peer reviewers.

For quantitative fields a “theorem-proof” result is the most convenient for establishing reproducibility because the theorem and the proof are both contained within the paper. However, important results are increasingly supported by the analysis of data, whether real-world data or data synthetically generated by (say) a simulation.

Standards for reproducibility of such results has lagged far behind our ability to collect, create, and analyze data, but there are now intense efforts to catch up. In the simulation community the *ACM Transactions on Modeling and Computer Simulation* or *TOMACS* (see https://dl.acm.org/journal/tomacs/) is one of the leaders in this effort.

Standards for reproducibility of results are evolving so quickly so that it is fruitless to give a specific prescription here. Instead we list a few basic principles that seem likely to remain relevant even as software, documentation, and archiving tools evolve:

Research principle 8. *Separate or distinguish the input data that defines each test case, scenario, or problem from the algorithms that generate simulated responses, and archive the input files that define each test case, scenario, or problem rather than altering a single input set.*

In research that employs simulation it is rare that there is only a single test case, scenario, or problem. Typically there are many that differ in systematic ways such as the number of systems K in ranking and selection, or the correlation in a randomly generated knapsack problem. Extreme (bad) practices that thwart reproducibility include creating a single piece of code that is edited for each test case, scenario or problem considered, or creating an entirely separate piece of code for each test case, scenario, or problem in which all inputs are "hard coded" into the algorithm.

Research principle 9. *The input data for a test case, scenario, or problem includes the random-number seeds or streams, and they should always be set rather than accepting software defaults.*

Nearly all software that has the capability to run stochastic simulations has a default setting for the random-number seeds or streams. However, these are not always fixed seeds or streams, and even when they are it is not always obvious what triggers them to be reset to their default starting values. One should always set and record the seeds or streams used.

Research principle 10. *Archive a script for describing how the experiments were executed and analyzed.*

Generation and analysis of data is not always a one-step process (see the next principle). Further, specific versions of some software may be required. You may need such a script to reproduce results yourself, and reviewers will certainly need it if asked to reproduce your results.

Research principle 11. *As far as possible, separate the data-generation aspect of the simulation from the analysis of it.*

If your experiments compare, say, the variance of control-variate estimators to sample means, then it is tempting to write code that generates the data, computes the estimators, computes the estimated variances, and saves only the final values.

Contrast this with generating all output data and retaining it; computing both estimators from the data and retaining them; and then computing the sample variances and retaining them. The latter approach facilitates asking different questions of the data at a later time, diagnosing where an error occurred if one is later suspected, as well as facilitating the examination of results by others without having to actually run the simulation.

In addition to facilitating reproducibility, practicing these research principles has a side effect of helping you find mistakes or bugs in your own work.

References

Albright, S. C. (2007). *VBA for modelers: Developing decision support systems using Microsoft Excel* (2nd ed.). Belmont: Thompson Higher Eduction.

Alexopoulos, C. (2006). Statistical estimation in computer simulation. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Andradóttir, S. (1999). Accelerating the convergence of random search methods for discrete stochastic optimization. *ACM Transactions on Modeling and Computer Simulation, 9*, 349–380.

Andradóttir, S. (2006a). An overview of simulation optimization via random search. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Andradóttir, S. (2006b). Simulation optimization with countably infinite feasible regions: Efficiency and convergence. *ACM Transactions on Modeling and Computer Simulation, 16*, 357–374.

Andradóttir, S., & Kim, S. (2010). Fully sequential procedures for comparing constrained systems via simulation. *Naval Research Logistics, 57*, 403–421.

Ankenman, B. E., Nelson, B. L. & Staum, J. (2010). Stochastic kriging for simulation metamodeling. *Operations Research, 58*, 371–382.

Ankenman, B. E., & Nelson, B. L. (2012). A quick assessment of input uncertainty. *Proceedings of the 2012 Winter Simulation Conference* (pp. 241–250). Piscataway, New Jersey: IEEE.

Asmussen, S. & Glynn, P. W., (2007). *Stochastic simulation: Algorithms and analysis*. New York: Springer.

Batur, D., & Kim, S. (2010). Finding feasible systems in the presence of constraints on multiple performance measures. *ACM Transactions on Modeling and Computer Simulation, 20*, 13:1–13:26.

Bechhofer, R. E., Santner, T. J., & Goldsman, D. (1995). *Design and analysis of experiments for statistical selection, screening and multiple comparisons*. New York: Wiley.

B. L. Nelson, L. Pei, *Foundations and Methods of Stochastic Simulation*, International Series in Operations Research & Management Science 316,
https://doi.org/10.1007/978-3-030-86194-0

Biller, B., & Corlu, C. G. (2012). Copula-based multivariate input modeling. *Surveys in Operations Research and Management Science, 17*, 69–84.

Biller, B., & Ghosh, S. (2006). Multivariate input processes. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Biller, B., & Nelson, B. L. (2003). Modeling and generating multivariate time-series input processes using a vector autoregressive technique. *ACM Transactions on Modeling and Computer Simulation, 13*, 211–237.

Biller, B., & Nelson, B. L. (2005). Fitting time series input processes for simulation. *Operations Research, 53*, 549–559.

Billingsley, P. (1995). *Probability and measure* (3rd ed.). New York: Wiley.

Boesel, J., Nelson, B. L., & Kim, S. (2003). Using ranking and selection to “clean up” after simulation optimization. *Operations Research, 51*, 814–825.

Bratley, P., Fox, B. L., & Schrage, L. E. (1987). *A guide to simulation* (2nd ed.). New York: Springer.

Bucklew, J. A. (2004). *Introduction to rare event simulation*. New York: Springer.

Burt, J. M., & Garman, M. B. (1971). Conditional Monte Carlo: A simulation technique for stochastic network analysis. *Management Science, 19*, 207–217.

Cario, M. C. & Nelson, B. L. (1998). Numerical methods for fitting and simulating autoregressive-to-anything processes. *INFORMS Journal on Computing, 10*, 72–81.

Cash, C., Nelson, B. L., Long, J., Dippold, D., & Pollard, W. (1992). Evaluation of tests for initial-condition bias. *Proceedings of the 1992 Winter Simulation Conference* (pp. 577–585). Piscataway, New Jersey: IEEE.

Chatfield, C. (2004). *The analysis of time series: An introduction* (6th ed.). Boca Raton: Chapman & Hall/CRC.

Chen, H. (2001). Initialization for NORTA: Generation of random vectors with specified marginals and correlations. *INFORMS Journal on Computing, 13*, 312–331.

Chen, H. & Schmeiser, B. W. (2019). MISE-optimal intervals for MNO-PQRS estimators of Poisson rate functions. *Proceedings of the 2019 Winter Simulation Conference* (pp. 368–379). Piscataway, New Jersey: IEEE.

Chen, Y., & Ryzhov, I. O. (2019). Complete expected improvement converges to an optimal budget allocation. *Advances in Applied Probability, 51*, 209–235.

Chow, Y. S., & Robbins, H. (1965). On the asymptotic theory of fixed-width sequential confidence intervals for the mean. *The Annals of Mathematical Statistics, 36*, 457–462.

Devroye, L. (1986). *Non-uniform random variate generation*. New York: Springer.

Devroye, L. (2006). Nonuniform random variate generation. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Dong, J., Feng, M. B., & Nelson, B. L. (2018). Unbiased metamodeling via likelihood ratios. *Proceedings of the 2018 Winter Simulation Conference* (pp. 1778–1789). Piscataway, New Jersey: IEEE.

Eckman, D. J., & Henderson, S. G. (2018). Guarantees on the probability of good selection. *Proceedings of the 2018 Winter Simulation Conference* (pp. 351–365). Piscataway, New Jersey: IEEE.

Eckman, D. J., & Henderson, S. G. (2018). Reusing search data in ranking and selection: What could possibly go wrong? *ACM Transactions on Modeling and Computer Simulation, 28*, 18:1–18:15.

Efron, B., & Tibshirani, R. J. (1993). *An introduction to the bootstrap*. Boca Raton: Chapman & Hall/CRC.

Elizandro, D., & Taha, H. (2008). *Simulation of industrial systems: Discrete event simulation using Excel/VBA*. New York: Auerbach Publications.

Fan, W., Hong, L. J., & Nelson, B. L. (2016). Indifference-zone-free selection of the best. *Operations Research, 64*, 1499–1514.

Frazier, P. I. (2010). Decision-theoretic foundations of simulation optimization. In J. J. Cochran (Ed.), *Wiley encyclopedia of operations research and management sciences*. New York: Wiley.

Frazier, P. I. (2018). A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811.

Frazier, P. I. (2014). A fully sequential elimination procedure for indifference-zone ranking and selection with tight bounds on probability of correct selection. *Operations Research, 62*, 926–942.

Fu, M. C. (2006). Gradient estimation. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Fu, M. C. (Ed.). (2015). *Handbook of simulation optimization*. New York: Springer.

Fujimoto, R. M. (2016). Research challenges in parallel and distributed simulation. *ACM Transactions on Modeling and Computer Simulation, 26*, 1–29.

Gerhardt, I., & Nelson, B. L. (2009). Transforming renewal processes for simulation of nonstationary arrival processes. *INFORMS Journal on Computing, 21*, 630–640.

Ghosh, S., & Henderson, S. G. (2002). Chessboard distributions and random vectors with specified marginals and covariance matrix. *Operations Research, 50*, 820–834.

Glasserman, P. (2004). *Monte Carlo methods in financial engineering*. New York: Springer.

Glasserman, P., & Yao, D. D. (1992). Some guidelines and guarantees for common random numbers. *Management Science, 38*, 884–908.

Glynn, P. W., & Heidelberger, P. (1991). Analysis of parallel replicated simulations under a completion time constraint. *ACM Transactions on Modeling and Computer Simulation, 1*, 3–23.

Glynn, P. W. (2006). Simulation algorithms for regenerative processes. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Glynn, P. W., & Iglehart, D. L. (1990). Simulation output analysis using standardized time series. *Mathematics of Operations Research, 15*, 1–16.

Glynn, P. & Juneja, S. (2004). A large deviations perspective on ordinal optimization. *Proceedings of the 2004 Winter Simulation Conference* (pp. 577–585). Piscataway, New Jersey: IEEE.

Glynn, P. W., & Whitt, W. (1992). The asymptotic validity of sequential stopping rules for stochastic simulations. *The Annals of Applied Probability, 2*, 180–198.

Goldsman, D., Kim, S., Marshall, S. W., & Nelson, B. L. (2002). Ranking and selection for steady-state simulation: Procedures and perspectives. *INFORMS Journal on Computing, 14*, 2–19.

Goldsman, D., & Nelson, B. L. (1998). Comparing systems via simulation. In J. Banks (Ed.), *Handbook of simulation* (pp. 273–306). New York: Wiley.

Goldsman, D., & Nelson, B. L. (2006). Correlation-based methods for output analysis. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Goyal, A., Heidelberger, P., & Shahabuddin, P. (1987). Measure specific dynamic importance sampling for availability simulations. In *Proceedings of the 1987 Winter Simulation Conference* (pp. 351-357). Piscataway, New Jersey: IEEE.

Gramacy, R.B. (2020). *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. Boca Raton FL: CRC Press.

Gross, D., Shortle, J. F., Thompson, J. M., & Harris, C. M. (2008). *Fundamentals of queueing theory* (4th ed.). New York: Wiley.

Haas, P. J. (2002). *Stochastic petri nets: Modeling, stability, simulation*. New York: Springer.

Heidelberger, P. (1988). Discrete event simulations and parallel processing: Statistical properties. *SIAM Journal on Scientific and Statistical Computing, 9*, 1114–1132.

Henderson, S. G. (2003). Estimation of nonhomogeneous Poisson processes from aggregated data. *Operations Research Letters, 31*, 375–382.

Henderson, S. G. (2006). Mathematics for simulation. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Henderson, S. G., & Nelson, B. L. (2006). Stochastic computer simulation. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Hill, R. R., & Reilly, C. H. (2000). The effects of coefficient correlation structure in two-dimensional knapsack problems on solution procedure performance. *Management Science, 46*, 302–317.

Hong, L. J., & Nelson, B. L. (2007a). Selecting the best system when systems are revealed sequentially. *IIE Transactions, 39*, 723–734.

Hong, L. J., & Nelson, B. L. (2007b). A framework for locally convergent random-search algorithms for discrete optimization via simulation. *ACM Transactions on Modeling and Computer Simulation, 17*, 19/1–19/22.

Hörmann, W. (1993). The transformed rejection method for generating Poisson random variables. *Insurance: Mathematics and Economics, 12*, 39–45.

Hunter, S. R., & Nelson, B. L. (2017). Parallel ranking and selection. In *Advances in Modeling and Simulation* (pp. 249–275). Springer, New York.

Iravani, S. M. R., & Krishnamurthy, V. (2007). Workforce agility in repair and maintenance environments. *Manufacturing and Service Operations Management, 9*, 168–184.

Iravani, S. M., Van Oyen, M. P., & Sims, K. T. (2005). Structural flexibility: A new perspective on the design of manufacturing and service operations. *Management Science, 51*, 151–166.

Jiang, W. X., Nelson, B. L., & Hong, L. J. 2019. Estimating sensitivity to input model variance. *Proceedings of the 2019 Winter Simulation Conference* (pp. 3705–3716). Piscataway, New Jersey: IEEE.

Johnson, M. E. (1987). *Multivariate statistical simulation*. New York: Wiley.

Johnson, N. L., Kemp, A. W., & Kotz, S. (2005). *Univariate discrete distributions* (3rd ed.). New York: Wiley.

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1994). *Continuous univariate distributions* (2nd ed., Vol. 1). New York: Wiley.

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1995). *Continuous univariate distributions* (2nd ed., Vol. 2). New York: Wiley.

Johnson, N. L., Kotz, S., & Balakrishnan, N. (1997). *Discrete multivariate distributions*. New York: Wiley.

Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization, 13*, 455–492.

Kachitvichyanukul, V., & Schmeiser, B. (1990). Noninverse correlation induction: Guidelines for algorithm development. *Journal of Computational and Applied Mathematics, 31*, 173–180.

Karian, A. Z., & Dudewicz, E. J. (2000). *Fitting statistical distributions: The generalized lambda distribution and generalized bootstrap methods*. New York: CRC.

Karlin, S., & Taylor, H. M. (1975). *A first course in stochastic processes* (2nd ed.). New York: Academic.

Kelton, W. D. (2006). Implementing representations of uncertainty. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Kelton, W. D., Smith, J. S., & Sturrock, D. T. (2011). *Simio and simulation: Modeling, analysis and applications* (2nd ed.). New York: McGraw-Hill.

Kim, S., & Nelson, B. L. (2001). A fully sequential procedure for indifference-zone selection in simulation. *ACM Transactions on Modeling and Computer Simulation, 11*, 251–273.

Kim, S., & Nelson, B. L. (2006). Selecting the best system. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Knuth, D. E. (1998). *The art of computer programming, Vol. 2: Seminumerical algorithms* (3rd ed.). Boston: Addison-Wesley.

Kotz, S., Balakrishnan, N., & Johnson, N. L. (2000). *Continuous multivariate distributions, Vol. 1, models and applications* (2nd ed.). New York: Wiley.

Kulkarni, V. G. (1995). *Modeling and analysis of stochastic systems*. London: Chapman & Hall.

Lakhany, A., & Mausser, H. (2000). Estimating the parameters of the generalized lambda distribution. *ALGO Research Quarterly, 3*, 47–58.

Lam, H. (2016). Advanced tutorial: Input uncertainty and robust analysis in stochastic simulation. *Proceedings of the 2016 Winter Simulation Conference* (pp. 178–192). Piscataway, New Jersey: IEEE.

Lam, H., & and Qian, H. (2018). Subsampling to enhance efficiency in input uncertainty quantification. 1811.04500, arXiv.

Law, A. M. (2007). *Simulation modeling and analysis* (4th ed.). New York: McGraw-Hill.

Law, A. M., & Kelton, W. D. (2000). *Simulation modeling and analysis* (3rd ed.). New York: McGraw-Hill.

L'Ecuyer, P. (1988). Efficient and portable combined random number generators. *Communications of the ACM, 31*, 742–749.

L'Ecuyer, P. (1990). A unified view of IPA, SF, and LR gradient estimation techniques. *Management Science, 36*, 1364–1383.

L'Ecuyer, P. (1999). Good parameters and implementations for combined multiple recursive random number generators. *Operations Research, 47*, 159–164.

L'Ecuyer, P. (2006). Uniform random number generation. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

L'Ecuyer, P., & Simard, R. (2001). On the performance of birthday spacings tests for certain families of random number generators. *Mathematics and Computers in Simulation, 55*, 131–137.

L'Ecuyer, P., Simard, R., Chen, E. J., & Kelton, W. D. (2002). An object-oriented random-number package with many long streams and substreams. *Operations Research, 50*, 1073–1075.

Lee, S., Wilson, J. R., & Crawford, M. M. (1991). Modeling and simulation of a nonhomogeneous Poisson process having cyclic behavior. *Communications in Statistics-Simulation and Computation, 20*, 777–809.

Leemis, L. M. (1991). Nonparameteric estimation of the cumulative intensity function for a nonhomogeneous Poisson process. *Management Science, 37*, 886–900.

Leemis, L. M. (2006). Arrival processes, random lifetimes and random objects. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Leemis, L. M., & McQueston, J. T. (2008). Univariate distribution relationships. *The American Statistician, 62*, 45–53.

Lehmann, E. L. (1999). *Elements of large-sample theory*. New York: Springer.

Louis, T. A. (1981). Confidence intervals for a binomial parameter after observing no successes. *The American Statistician, 35*, 154.

Luo, J., Hong, L. J., Nelson, B. L., & Wu, Y. (2015). Fully sequential procedures for large-scale ranking-and-selection problems in parallel computing environments. *Operations Research, 63*, 1177–1194.

Marse, K., & Roberts, S. D. (1983). Implementing a portable FORTRAN uniform (0,1) generator. *Simulation, 41*, 135–139.

Montgomery, D. C. (2009). *Design and analysis of experiments* (7th ed.). New York: Wiley.

Morgan, L. E., Nelson, B. L., Titman, A. C. & Worthington, D. J. (2019). A spline-based method for modelling and generating a nonhomogeneous Poisson process. *Proceedings of the 2019 Winter Simulation Conference* (pp. 356–367). Piscataway, New Jersey: IEEE.

Nádas, A. (1969). An extension of a theorem of Chow and Robbins on sequential confidence intervals for the mean. *The Annals of Mathematical Statistics, 40*, 667–671.

Nelson, B. L., & Leemis, L. M. (2020). The ease of fitting but futility of testing a nonstationary Poisson processes from one sample path. *Proceedings of the 2020 Winter Simulation Conference* (pp. 266–276). Piscataway, New Jersey: IEEE.

Nelson, B. L. (1990). Control-variate remedies. *Operations Research, 38*, 974–992.

Nelson, B. L. (1995). *Stochastic modeling: Analysis and simulation*. Mineola: Dover Publications, Inc.

Nelson, B. L. (2008). The MORE plot: Displaying measures of risk and error from simulation output. *Proceedings of the 2008 Winter Simulation Conference* (pp. 413–416). Piscataway, New Jersey: IEEE.

Nelson, B. L., & Taaffe, M. R. (2004). The $Ph_t/Ph_t/\infty$ queueing system: Part I: The single node. *INFORMS Journal on Computing, 16*, 266–274.

Nelson, B. L., Swann, J., Goldsman, D., & Song, W. (2001). Simple procedures for selecting the best simulated system when the number of alternatives is large. *Operations Research, 49*, 950–963.

Owen, A. B. (2019). *Monte Carlo theory, methods and examples*. http://statweb.stanford.edu/~owen/mc/ [last accessed May 23, 2021].

Pasupathy, R., & Schmeiser, B. (2010). The initial transient in steady-state point estimation: Contexts, a bibliography, the MSE criterion, and the MSER statistic. *Proceedings of the 2010 Winter Simulation Conference* (pp. 184–197). Piscataway, New Jersey: IEEE.

Pei, L., Nelson, B. L., & Hunter, S. R. (2020). Evaluation of bi-Pass for parallel simulation optimization. *Proceedings of the 2020 Winter Simulation Conference* (pp. 2960–2971). Piscataway, New Jersey: IEEE.

Robinson, R. (2014) *Simulation: The practice of model development and use*. London: Red Globe Press.

Salemi, P., Song, E., Nelson, B. L., & Staum, J. (2019). Gaussian Markov random fields for discrete optimization via simulation: Framework and algorithms. *Operations Research, 67*, 250–266.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M. & Tarantola, S. (2008). *Global sensitivity analysis: The primer*. New York: John Wiley & Sons.

Sanchez, S. M., Wan, H., & Lucas, T. W. 2009. Two-phase screening procedure for simulation experiments. *ACM Transactions on Modeling and Computer Simulation, 19*, 1–24.

Santner, T. J., Williams, B. J. & Notz, W. I. (2003). *The design and analysis of computer experiments*. New York: Springer.

Sargent, R. G. (2011). Verification and validation of simulation models. *Proceedings of the 2011 Winter Simulation Conference* (pp. 183–198). Piscataway, New Jersey: IEEE.

Schmeiser, B. (1982). Batch size effects in the analysis of simulation output. *Operations Research, 30*, 556–568.

Schruben, L. (1982). Detecting initialization bias in simulation output. *Operations Research, 30*, 569–590.

Schruben, L., Singh, H., & Tierney, L. (1983). Optimal tests for initialization bias in simulation output. *Operations Research, 31*, 1167–1178.

Shapiro, A., Dentcheva, D., & Ruszczyński, A. (2009). *Lectures on stochastic programming: Modeling and theory*. Philadelphia: Society for Industrial and Applied Mathematics.

Shechter, S. M., Schaefer, A. J., Braithwaite, R. S., & Roberts, M. S. (2006). Increasing the efficiency of Monte Carlo cohort simulations with variance reduction techniques. *Medical Decision Making, 26*, 550–553.

Snell, M., & Schruben, L. (1985). Weighting simulation data to reduce initialization effects. *IIE Transactions, 17*, 354–363.

Song, E., & Nelson, B. L. (2015). Quickly assessing contributions to input uncertainty. *IIE Transactions, 47*, 893–909.

Song, E., & Nelson, B. L. (2017). Input model risk. In *Advances in Modeling and Simulation* (pp. 63–80). New York: Springer.

Steiger, N. M., & Wilson, J. R. (2001). Convergence properties of the batch means method for simulation output analysis. *INFORMS Journal on Computing, 13*, 277–293.

Stigler, S. M. (1986). *The history of statistics: The measurement of uncertainty before 1900*. Cambridge, MA: Belknap.

Süli, E., & Mayers, D. F. (2003). *An introduction to numerical analysis*. Cambridge University Press.

Swain, J. J., Venkatraman, S., & Wilson, J. R. (1988). Least-squares estimation of distribution functions in Johnson's translation system. *Journal of Statistical Computation and Simulation, 29*, 271–297.

Tafazzoli, A., & Wilson, J. R. (2011). Skart: A skewness-and-autoregression-adjusted batch-means procedure for simulation analysis. *IIE Transactions, 43*, 110–128.

Tamhane, A. C. 2009. *Statistical analysis of designed experiments: Theory and applications*. New York: John Wiley & Sons.

Walkenbach, J. (2010). *Excel 2010 power programming with VBA*. New York: Wiley.

White, K. P. (1997). An effective truncation heuristic for bias reduction in simulation output. *Simulation, 69*, 323–334.

Whitt, W. (1981). Approximating a point process by a renewal process: The view through a queue, an indirect approach. *Management Science, 27*, 619–636.

Whitt, W. (1989). Planning queueing simulations. *Management Science, 35*, 1341–1366.

Whitt, W. (2006). Analysis for design. In S. G. Henderson & B. L. Nelson (Eds.), *Handbooks in operations research and management science: Simulation*. New York: North-Holland.

Whitt, W. (2007). What you should know about queueing models to set staffing requirements in service systems. *Naval Research Logistics, 54*, 476–484.

Wieland, J. R., & Schmeiser, B. W. 2006. Stochastic gradient estimation using a single design point. *Proceedings of the 2006 Winter Simulation Conference* (pp. 390–397). Piscataway, New Jersey: IEEE.

Xu, J., Hong, L. J., & Nelson, B. L. (2010). Industrial strength COMPASS: A comprehensive algorithm and software for optimization via simulation. *ACM Transactions on Modeling and Computer Simulation, 20*, 1–29.

Xu, J., Nelson, B. L., & Hong, L. J. (2013). An adaptive hyperbox algorithm for high-dimensional discrete optimization via simulation problems. *INFORMS Journal on Computing, 25*, 133–146.

Zheng, Z. & Glynn, P. W. (2017). Fitting continuous piecewise linear Poisson intensities via maximum likelihood and least squares. *Proceedings of the 2017 Winter Simulation Conference* (pp. 1740–1749). Piscataway, New Jersey: IEEE.

Index