

## Chapter 2

# SIMULATION BASICS

### 1. Chapter Overview

This chapter has been written to introduce the topic of **discrete-event** simulation. To comprehend the material presented in this chapter, some background in the theory of probability is needed, some of which is in the Appendix. Two of the main topics covered in this chapter are random number generation and simulation modeling of random systems. Readers familiar with this material can skip this chapter without loss of continuity.

### 2. Introduction

A system is usually defined as a collection of entities that interact with each other. A simple example is the queue that forms in front of a teller in a bank: see Fig. 2.1. The entities in this system are the people (customers), who arrive to get served, and the teller (server), who provides service.

The *behavior* of a system can be described in terms of the so-called *state* of the system. In our queuing system, one possible definition for the system state is the length of the queue, i.e., the number of people waiting in the queue. Frequently, we are interested in how a system changes over time, i.e., how the state changes as time passes. In a real banking queue, the queue length fluctuates with time, and thus the behavior of the system can also be said to change with time. Systems which change their state with time are called *dynamic* systems. In this book, we will restrict our attention to discrete-event systems. In a

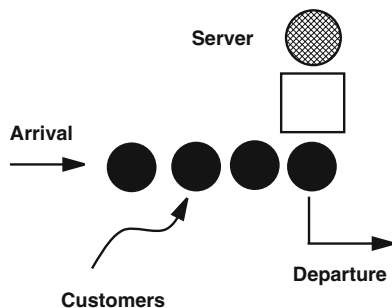


Figure 2.1. A schematic of a queue in a bank

**discrete-event** system, the time interval between two successive state changes, or alternatively two successive “events,” is of a finite duration. Hence, a system will be understood to be a discrete-event system throughout this book.

The behavior of any dynamic system is usually *governed* by some *variables*, often called the **governing variables**. The governing variables in our banking system are the time taken by the teller to provide service to a customer and the time between successive arrivals of customers to the bank. An appropriate question at this point is: Why should these two quantities be considered to be the governing variables? The answer is that using laws from queuing theory, one can show that the behavior of the system (e.g., the queue length) depends on the values assumed by these variables.

When the governing variables are random variables, the system is referred to as a *random* or *stochastic system*. Therefore, the reader should always keep in mind that a random system’s behavior is governed by one or more *random* variables. Knowledge of the distributions of these random variables is needed to analyze the system’s behavior.

### 3. Models

To understand, to analyze, and to predict the behavior of systems (both random and deterministic), operations researchers construct **models**. These models are typically *abstract* models, unlike physical models, e.g., a miniature airplane. Abstract models take the form of equations, functions, inequations (inequalities), and computer programs etc. To understand how useful abstract models can be, consider the simple model from Newtonian physics:  $v = u + gt$ . This equation *predicts* the speed of a freely-falling body that has been in the air for  $t$  time units after starting its descent at a speed of  $u$ .

The literature in stochastic operations research is full of mathematical models similar to that above which help in analyzing and predicting behavior of stochastic systems. Hence, stochastic operations research is sometimes described as the **physics** of stochastic systems. Queuing theory, renewal theory, and Brownian motion theory have been exploited to construct powerful mathematical models.

Although these mathematical models enjoy an important place in operations research, they are often tied to assumptions made about the system—assumptions that are sometimes necessary to develop a model. These assumptions may be related to the system's structure or to the distributions of the governing random variables. For instance, many queuing-theory models are limited in use to exponentially distributed service and inter-arrival times. Some models related to Brownian motion are restricted to the so-called heavy traffic conditions. Not surprisingly, stochastic operations research has always found models that generalize beyond such narrow assumptions to be very attractive.

One possible avenue for generating powerful models for large and complex systems is via the use of a computer program that mimics the system's behavior. This computer program is called a *simulation model*. The program usually achieves its goal of behavior prediction by generating random numbers for the governing random variables.

For a long time, simulation models did not receive the respect that they deserve. A primary reason was that although they could be used for analyzing random systems, their use in **optimizing** systems was not well understood or as well evolved as it is today. On the other hand, mathematical models, among other things such as being elegant, can usually optimize systems. Fortunately, things have changed, and now simulation models too can be used for optimizing systems. Furthermore, they can be used to optimize *complex, large-scale, and stochastic systems* for which it may be difficult to construct mathematical models.

We note that in this book, we are keenly interested in being able to *optimize* or *control* the stochastic system, so that it operates efficiently and/or the net costs (revenues) of running the system are reduced (increased). And we will use simulation as a tool to measure the system's performance (efficiency, costs, or revenues). However, in order to achieve this goal, it is important for the reader to gain a thorough understanding of the fundamental principles underlying discrete-event simulation. In what follows, we have made a serious attempt to explain the inner workings of a simulation model.

## 4. Simulation Modeling

Determining the distributions of the governing random variables is the first step towards modeling a stochastic system, regardless of whether mathematical or simulation models are used. In mathematical models, the *pdfs* (or *cdfs*) of the governing random variables are used in the closed forms obtained. In simulation models, the *pdfs* (or *cdfs*) are used to generate random numbers for the variables concerned. These random numbers are then used to imitate, within a computer, the behavior of the system. Imitating the behavior of a system essentially means *re-creating* the events that occur in the real-world system that is being imitated.

How does one determine the distribution of a random variable? For this, usually, one has to actually collect data on the values of the random variables from the real-life system. Then, from this data, it is usually possible to fit a distribution to that data, which is called distribution fitting. For a good discussion on distribution fitting, see e.g., [188].

An important issue in stochastic analysis is related to the *number* of random variables in the system. Generally, the larger the number of governing random variables in a system the more complicated is its analysis. This is especially true of analysis with mathematical models. On the other hand, *simulating* a system with several governing random variables has become a trivial task with modern-day simulation packages.

Our main strategy in simulation is to *re-create*, within a computer program, the events that take place in the real-life system. The re-creation of events is based on using suitable values for the governing random variables. For this, one needs a mechanism for generating values of the governing random variables. We will first discuss how to create random values for these variables and then discuss how to use the random values to re-create the events.

### 4.1. Random Number Generation

Here, we will discuss some popular random number generation schemes. We begin with a scheme for the uniform distribution between 0 and 1.

#### 4.1.1 Uniform Distribution in $(0, 1)$

We must make it clear at the outset that the random numbers that we will discuss here are *artificial*. “True” random numbers cannot be generated by a computer, but must be generated by a human brain

or obtained from a real system. Having said that, for all practical purposes, *artificial* (or pseudo) random numbers generated by computers are usually sufficient in simulations. Artificial random number generation schemes are required to satisfy some statistical tests in order to be acceptable. Needless to add, the schemes that we have at our disposal today do pass these tests. We will discuss one such scheme.

The so-called *linear congruential generator* of random numbers [221] is given by the following equation:

$$I_{j+1} \leftarrow (aI_j \bmod m), \quad j = 0, 1, 2, \dots \quad (2.1)$$

where  $a$  and  $m$  are positive integers. Equation (2.1) should be read as follows: the *remainder* obtained after  $aI_j$  is divided by  $m$  is denoted by  $(aI_j \bmod m)$ . The equation above provides an iterative scheme in which if one sets a positive value less than or equal to  $m$  for  $I_0$ , the sequence of values  $I_1, I_2, I_3, \dots$  will produce integers between 0 and  $m$ , where both 0 and  $m$  excluded.

To illustrate this idea, consider the following example. Let  $a = 2$ ,  $m = 20$  and  $I_0 = 12$ . (Please note this set of values for  $a$  and  $m$  are used only for illustration and are not recommended in practice.) The sequence that will be generated is:

$$(12, 4, 8, 16, 12, 4, 8, 16, 12, 4, 8, 16, \dots) \quad (2.2)$$

This sequence has integers between 1 and  $m - 1 = 19$ , both 1 and 19 included. It cannot include 20 or any integer greater than 20, because each integer is obtained after a division by 20. Then, we can conclude that in the sequence defined in (2.2), the terms

$$12, 4, 8, 16$$

form a set that has integers ranging from 0 to 20 (both 0 and 20 excluded).

By a suitable choice of  $a$  and  $m$ , it is possible to generate a set that contains *all* the different integers in the range from 0 to  $m$  and each integer appears only once in the set. The number of elements in such a set will be  $m - 1$ . If each integer in this set is divided by  $m$ , a set of numbers in the interval  $(0, 1)$  will be obtained. In general, if the  $i$ th integer is denoted by  $x_i$  and the  $i$ th random number is denoted by  $y_i$ , then

$$y_i = x_i/m.$$

Now, each number ( $y_i$ ) in this set will be equally likely, because each associated integer ( $x_i$ ), between 0 and  $m$ , occurs *once* at some point

in the original set. Then, for large values of  $m$ , this set of random numbers from 0 to 1 will *approximate* a set of natural random numbers from the uniform distribution. Recall that the *pdf* of the random variable has the same value at every point in the uniform distribution.

The *maximum* number of integers that may be generated in this process before it starts repeating itself is  $m - 1$ . Also, if  $I_0$ , which is known as the **seed**, equals 0, the sequence will only contain zeroes. A suitable choice of  $a$  and  $m$  yields a set of  $m - 1$  numbers such that each integer between 0 and  $m$  occurs once at some point.

An important question is: Is it acceptable to use a sequence of random numbers with repeating subsets, e.g.,  $(12, 4, 8, 16, 12, 4, 8, 16, \dots)$ ? The answer is no because the numbers  $(12, 4, 8, 16)$  repeat and are therefore deterministic. This is a serious problem. Now, unfortunately, random numbers from the linear congruential generator must repeat after a *finite* period. Therefore, the only way out of this is to generate a sequence that has a *sufficiently long* period, such that we are finished with using the sequence before it gets to repeats itself. Fortunately, if  $m = 2^{31} - 1$ , then with a suitable choice of  $a$ , it is possible to generate a sequence that has a period of  $m - 1$ . Thus, if the number of random numbers needed is less than  $m - 1 = 2,147,483,646$  (for most applications, this is sufficient), we have a set of random numbers with no repetitions.

Schemes with small periods produce erroneous results. Furthermore, repeating numbers are also not *independent*. In fact, repetitions imply that the numbers stray far away from the uniform distribution that they seek to approximate.

If the largest number in a computer's memory is  $2^{31} - 1$ , then a legal value for  $a$  that goes with  $m = 2^{31} - 1$  is 16,807. These values of  $m$  and  $a$  **cannot** be implemented naively in the computer program. The reason is easy to see. In the multiplication of  $a$  by  $I_j$ , where the latter can be of the order of  $m$ , one runs into trouble as the product is often larger than  $m$ , the largest number that the computer can store. A clever trick from Schrage [266] helps us circumvent this difficulty. Let  $[x/y]$  denote the *integer* part of the quotient obtained after dividing  $x$  by  $y$ . Using Schrage's approximate factorization, if

$$Q = a(I_j \bmod q) - r[I_j/q],$$

the random number generation scheme is given by

$$I_{j+1} \leftarrow \begin{cases} Q & \text{if } Q \geq 0 \\ Q + m & \text{otherwise.} \end{cases}$$

In the above,  $q$  and  $r$  are positive numbers. As is clear from this *approximate* factorization, multiplication of  $a$  and  $I_j$ , which is required in Eq. (2.1), is avoided. For  $a = 7^5 = 16,807$  and  $m = 2^{31} - 1$ , values suggested for  $q$  and  $r$  are:  $q = 127,773$  and  $r = 2,836$ . Other values can be found in e.g., [221]. The period of this scheme is  $m - 1 = 2^{31} - 2$ . When the number of calls to this scheme becomes of the order of the period, it starts repeating numbers and is **not** recommended. Two sequences of different periods can be combined to give a sequence of a longer period [189]. The reader is referred to [189] for further reading. See [239] for useful computer programs.

The above-described scheme for generating random numbers from the uniform distribution  $(0, 1)$  forms the work horse for random number generation in many commercial packages. It can be used for generating random numbers from any other distribution.

#### 4.1.2 Inverse Function Method

In this subsection, we will discuss how random numbers for some distributions can be generated. In particular, we will discuss the inverse function method.

The inverse function method relies on manipulating the *cdf* of a function. This often requires the *cdf* to be “nice.” In other words, the *cdf* should be of a form that can be manipulated. We will show how this method works on the uniform distribution  $(a, b)$  and the exponential distribution.

For any given value of  $x$ , the *cdf* has to assume a value between 0 and 1. Conversely, it may be said that when  $F(x)$  assumes a value between 0 and 1, that particular value corresponds to some value of  $x$ . Hence, one way to find a random number from a given distribution is to consider a *random* value for  $F(x)$ , say  $y$ , and then determine the value of  $x$  that corresponds to  $y$ . Since values of  $F(x)$  lie between 0 and 1, we can assume that  $y$  must come from the uniform distribution  $(0, 1)$ . Hence, our strategy in the inverse function method is to generate a random number  $y$  from the uniform distribution  $(0, 1)$ , equate it to the *cdf*, and then solve for  $x$ , which will denote the random number we desire.

The *cdf* of the uniform distribution  $(a, b)$  is given by

$$F(x) = \frac{x - a}{b - a}.$$

Hence, using our strategy and solving for  $x$ , we have:

$$y = F(x) = \frac{x - a}{b - a}, \text{ i.e., } x = a + y(b - a).$$

The *cdf* of the exponential distribution is given by:

$$F(x) = 1 - e^{-\lambda x}, \text{ where } \lambda > 0.$$

Then using the same strategy, we obtain:

$$F(x) = y, \text{ i.e., } 1 - e^{-\lambda x} = y, \text{ i.e., } e^{-\lambda x} = 1 - y.$$

Taking the natural logarithm of both sides, after some simplification, we have:

$$x = -\frac{\ln(1 - y)}{\lambda}.$$

Replacing  $(1 - y)$  by  $y$  leads to  $x = -\frac{\ln(y)}{\lambda}$ , which is also an acceptable rule because  $y$  is also a number between 0 and 1.

The method described above is called the inverse function method essentially because the *cdf* is manipulable and it is possible to solve for  $x$  for a given value of  $F(x)$ . Sometimes the closed form for the *cdf* may not be manipulable directly, but it may be possible to develop a closed-form approximation of the inverse of the *cdf*. Such approximations are often acceptable in computer simulation, because even computing the logarithm in the scheme described above for the exponential distribution requires an approximation. A remarkable example of this is the closed form approximation of the inverse of the normal distribution's *cdf*, due to Schmeiser [263], which leads to the following formula for generating a random number from the *standard* normal distribution:

$$x = \frac{y^{0.135} - (1 - y)^{0.135}}{0.1975}.$$

This provides one decimal-place accuracy for  $0.0013499 \leq y \leq 0.9986501$  at the very least. From this, one can determine a random number for the normal distribution having a mean of  $\mu$  and a standard deviation of  $\sigma$  using:  $\mu + x\sigma$ .

Clearly, the inverse function method breaks down in the absence of a manipulable closed form for the *cdf* or an approximation for the inverse *cdf*. Then, one must use other methods, e.g., the acceptance-rejection method, which is discussed in numerous textbooks [188].

We will now turn to an important mechanism lying at the heart of discrete-event computer simulation. A clear understanding of it is vitally essential.

## 4.2. Event Generation

The best way to explain how values for random variables can be used to re-create events within a simulator is to use an example. We will



use the single-server queuing example that has been discussed above. In the single-server queue, there are two governing variables, both of which are random:

1. The time between successive arrivals to the system:  $t_a$
2. The time taken by the server to give service to one customer:  $t_s$ .

We now generate values for the elements of the two sequences. Since we know the distributions, it is possible to generate values for them. E.g.,

let the first 7 values for  $t_a$  be: 10.1, 2.3, 1, 0.9, 3.5, 1.2, 6.4

and those for  $t_s$  be: 0.1, 3.2, 1.19, 4.9, 1.1, 1.7, 1.5.

Now  $\{t(n)\}_{n=1}^{\infty}$  will denote the following sequence:  $\{t(1), t(2), \dots\}$ . These values, we will show below, will lead to re-enacting the real-life queue.

If one observes the queue from real life and collects data for any of these sequences from there, the values obtained may not necessarily be identical to those shown above. Then, how will the above lead to a re-enactment of the real-life system within our simulation model? The answer is that the elements of this sequence belong to the distribution of the random variable in the real-life system. In other words, the above sequence could very well be a sequence from the real-life system. We will discuss later why this is *sufficient* for our goals in simulation modeling.

Now, from the two sequences, one can construct a sequence of the events that occur. The events here are of two types:

1. A customer enters the system (**arrival**).
2. A customer is serviced and leaves the system (**departure**).

Our task of re-creating events boils down to the task of finding the time of occurrence of each event. In the single-server queuing system (see Fig. 2.2), when a customer arrives to find that the server is idle, he or she directly goes to the server without waiting. An arriving customer who finds that the server is busy providing service to someone becomes either the first person in the queue or joins the queue's end. The arrivals in this case, we will assume, occur regardless of the number of people waiting in the queue.

To analyze the behavior of our system, the first task is to determine the clock time of each event as it occurs. This task, as stated above,

lies at the heart of stochastic simulation. If one can accomplish this task, various aspects of the system can be analyzed.

Before delving into the details of how we can accomplish this task, we need a clear understanding of what the elements of the sequences  $\{t_a(n)\}_{n=1}^{\infty}$  and  $\{t_s(n)\}_{n=1}^{\infty}$  stand for. The first element in each sequence is associated with the first customer to arrive in the system, the second is associated with the second customer, and so on. Keeping this in mind, note that in the queuing example, there are two types of events: arrivals and departures. Our task in each case can be accomplished as described below:

1. For arrivals: The clock time of any arrival in the system will be equal (in this case) to (1) the clock time of the previous arrival plus (2) the inter-arrival time of the current arrival. (Remember the second quantity is easy to find. The inter-arrival time of the  $k$ th arrival is  $t_a(k)$ .)
2. For departures:
  - a. If an arriving customer finds that the server is idle, the next departure will occur at a clock time equal to the arrival clock time of the arriving customer plus the service time of the arriving customer. (Again, the second quantity is easy to find. The service time of the  $k$ th arrival is  $t_s(k)$ .)
  - b. If an arriving customer finds that the server is busy, the next departure will occur at a time equal to (1) the clock time of the *previous* departure plus (2) the service time of the customer *currently being served*.

We now illustrate these ideas with the values generated for the sequence. See Fig. 2.2. The first arrival takes place at clock time of 10.1, the second at clock time of  $10.1 + 2.3 = 12.4$ , the third at clock time of  $12.4 + 1 = 13.4$ , the fourth at clock time of  $13.4 + 0.9 = 14.3$ , the fifth at a clock time of  $14.3 + 3.5 = 17.8$ , the sixth at a clock time of  $17.8 + 1.2 = 19$ , and the seventh at a clock time of  $19 + 6.4 = 25.4$ .

When the first arrival occurs, there is nobody in the queue and hence the first departure occurs at a clock time of  $10.1 + 0.1$  (service time of the first customer) = 10.2. Now, from the clock time of 10.2 till the clock strikes 12.4, when the second arrival occurs, the server is idle. So when the second arrival occurs, there is nobody in the queue and hence the time of the second departure is  $12.4 + 3.2$  (the service time of the second arrival) = 15.6. The third arrival takes place at a clock time of 13.4, but the second customer departs much later at a clock time of

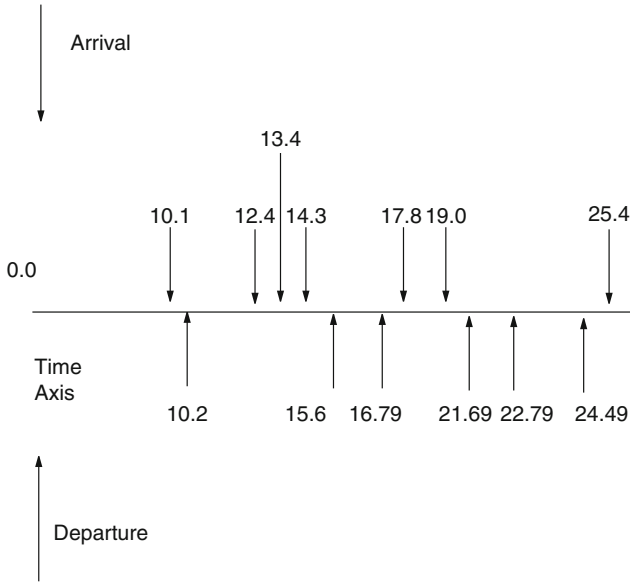


Figure 2.2. The event clock showing arrivals and departures

15.6. Hence the second customer must wait in the queue till 15.6 when he/she joins service. Hence the third departure will occur at a clock time of 15.6 plus the service time of the third customer, which is 1.19. Therefore, the departure of the third customer will occur at a clock time of  $15.6 + 1.19 = 16.79$ . The fourth customer arrives at a clock time of 14.3 but the third departs only at 16.79. It is clear that the fourth customer will depart at a clock time of  $16.79 + 4.9 = 21.69$ . In this way, we can find that the fifth departure will occur at a clock time of 22.79 and the sixth at a clock time of 24.49. The seventh arrival occurs at a clock time of 25.4, which is after the sixth customer has departed. Hence when the seventh customer enters the system, there is nobody in the queue, and the seventh customer departs some time after the clock time of 25.4. We will analyze the system until the time when the clock strikes 25.4.

Now, from the sequence of events constructed, we can collect data related to system parameters of interest. First consider server utilization. From the observations above, it is clear that the server was idle from a clock time of 0 until the clock struck 10.1, i.e., for a *time interval* of 10.1. Then again it was idle from the time 10.2 (the clock time of the first departure) until the second arrival at a clock time of 12.4, i.e., for 2.2 time units. Finally, it was idle from the time 24.49

(the clock time of the sixth departure) until the seventh arrival at a clock time of 25.4, i.e., for 0.91 time units. Thus, based on our observations, we can state that the system was idle for a total time of  $10.1 + 2.2 + 0.91 = 13.21$  out of the total time of 25.4 time units for which we observed the system. Then, the server utilization (fraction of time for which the server was busy) is clearly:  $1 - \frac{13.21}{25.4} = 0.4799$ .

If one were to create very *long* sequences for the inter-arrival times and the service times, one could then obtain estimates of the utilization of the server *over a long run*. Of course, this kind of a task should be left to the computer, but the point is that computer programs are thus able to collect estimates of parameters measured over a long run.

It should now be clear to the reader that although the sequences generated may not be *identical* to sequences obtained from actual observation of the original system, what we are really interested in are the estimates of system parameters, e.g., long-run utilization. As long as the sequence of values for the governing random variables are generated from the correct distributions, reliable estimates of these parameters can be obtained. For instance, an estimate like long-run utilization will approach a constant as the simulation horizon (25.4 time units in our example) approaches infinity.

Many other parameters for the queuing system can also be measured similarly. Let  $E[W]$  denote the average customer waiting time in the queue. Intuitively, it follows that the average waiting time can be found by summing the waiting times of a large number (read infinity) of customers and dividing the sum by the number of customers. Thus if  $w_i$  denotes the queue waiting time of the  $i$ th customer, the long-run average waiting time should be

$$E[W] = \lim_{n \rightarrow \infty} \frac{\sum_{i=1}^n w_i}{n}. \quad (2.3)$$

Similarly, the *long-run* average number in the queue can be defined as:

$$E[Q] = \lim_{T \rightarrow \infty} \frac{\int_0^T Q(t) dt}{T}, \quad (2.4)$$

where  $Q(t)$  denotes the number in the queue at time  $t$ . Now, to use these definitions, in practice, one must treat  $\infty$  as a large number.

For *estimating* the long-run average waiting time, we can use the following formula:

$$\tilde{W} = \frac{\sum_{i=1}^n w_i}{n}, \quad (2.5)$$

where  $\tilde{W}$  is essentially the sample mean from  $n$  samples. In the example of Fig. 2.2,

$$w_1=w_2=0, w_3=15.6 - 13.4, w_4 = 16.79 - 14.3, w_5 = 21.69 - 17.8, \text{ and} \\ w_6 = 22.79 - 19.0, \text{ i.e., } \tilde{W} = \frac{0 + 0 + 2.2 + 2.49 + 3.89 + 3.79}{6} = 2.06$$

For estimating the average number in the queue, one uses a similar mechanism. The formula for the estimate is:

$$\tilde{Q} = \frac{\sum_{i=1}^n t_i Q_i}{T}, \quad (2.6)$$

where  $t_i$  is the time duration for which there were  $Q_i$  customers in the queue, and  $T = \sum_{i=1}^n t_i$  is the amount of time for which simulation is conducted. Again,  $\tilde{Q}$  is a sample mean.

### 4.3. Independence of Samples Collected

Often, in simulation analysis, one runs the simulation a number of times, using a *different* set of random numbers in each run. By changing the seed, defined above, one can generate a new set of random numbers. Each run of the simulation with a given set of random numbers is called a **replication**. One replication yields only one *independent* sample for the quantity (or quantities) we are trying to estimate. The reason for this independence is that each replication runs with a *unique* set of random numbers, and hence, an estimate from one replication is independent of that from another. On the other hand, samples from within the same replication may depend on each other, and thus they are not independent. Independent estimates provide us with a mechanism to estimate the true mean (or variance) of the parameter of interest. Independent estimates also allow us to construct confidence intervals on the estimates of means obtained from simulations. The need for independence follows from the strong law of large numbers that we discuss below.

In practice, one estimates sample means from numerous replications, and the means from all the replications are averaged to obtain a *reliable* estimate of the true mean. This process of estimating the mean from several replications is called the *independent replications* method. We can generalize the concept as follows. Let  $\tilde{W}_i$  denote the estimate of the waiting time from  $n$  samples from the  $i$ th replication. Then, a good estimate of the true mean of the waiting time from  $k$  replications can be found from the following:

$$E[W] = \frac{\sum_{i=1}^k \tilde{W}_i}{k},$$

provided  $k$  is large enough.

Averaging over many independent estimates (obtained from many replications), therefore, provides a good estimate for the true mean of the parameter we are interested in. However, doing multiple replications is not the only way to generate independent samples. There are other methods, e.g., the **batch means** method, which uses one *long* replication. The batch means method is a very intelligent method (see Schmeiser [264]) that divides the output data from a long replication into a small number of large batches, after deletion of some data. The means of these batches, it can be shown, can be treated as independent samples. These samples are then used to estimate the mean. We now present the mathematical result that allows us to perform statistical computations from means.

**THEOREM 2.1** (*The Strong Law of Large Numbers*) *Let  $X_1, X_2, \dots$  be a sequence (a set with infinitely many elements) of independent random variables having a common distribution with mean  $E(X) = \mu$ . Then, with probability 1,*

$$\lim_{k \rightarrow \infty} \frac{X_1 + X_2 + X_3 + \dots + X_k}{k} = \mu.$$

In the above theorem,  $X_1, X_2, X_3, \dots$  can be viewed as values of the same random variable from a given distribution. Essentially, what the theorem implies is that if we draw  $k$  independent samples of the random variable, then the sample mean, i.e.,  $\frac{X_1 + X_2 + X_3 + \dots + X_k}{k}$ , will tend to the actual mean of the random variable as  $k$  becomes very large.

This is an important result used heavily in simulations and in many other settings where samples are obtained from populations to make predictions about the population. While what this law states may be intuitively obvious, what one needs to remember is that the samples drawn have to be independent. Its proof can be found in [252] under some assumptions on the second moment of the random variable.

It should now be clear to the reader why the means obtained from a replication or a batch must be used to estimate the mean in simulations. The independence of the estimates obtained in simulation (either from replications or batches) also allows us to determine confidence intervals (of the mean) and prediction intervals of the parameter of interest. Estimating means, variances, and confidence

and prediction intervals is an important topic that the reader should become familiar with.

## 5. Concluding Remarks

Simulation of dynamic systems using random numbers was the main topic covered in this chapter. However, this book is not about the methodology of simulation, and hence our discussion was not comprehensive. Nevertheless, it is an important topic in the context of simulation-based optimization, and the reader is strongly urged to get a clear understanding of it. A detailed discussion on writing simulation programs in C can be found in [234, 188]. For an in-depth discussion on tests for random numbers, see Knuth [177].

**Historical Remarks:** The earliest reference to generating random numbers can be traced to George Louis Leclerc (later called Buffon) in 1733. It was in the 1940s, however, that “Monte Carlo simulation” first became known, and it was originally used to describe random number generation from a distribution. However, the name is now used loosely to refer to any simulation (including discrete-event) that uses random numbers. The advent of computers during the 1960s led to the birth of *computer* simulation. The power of computers has increased dramatically in the last few decades, enabling it to play a major role in analyzing stochastic systems. The fundamental contributions in simulation were made by pioneers in the industrial engineering community, who worked tirelessly through the decades allowing it to develop into a reliable and sophisticated science that it is today [14].

Simulation-Based Optimization

Parametric Optimization Techniques and Reinforcement  
Learning

Gosavi, A.

2015, XXVI, 508 p. 42 illus., Hardcover

ISBN: 978-1-4899-7490-7