

As discussed in Chapter 1, `ssq1` and `sis1` are examples of trace-driven discrete-event simulation programs. By definition, a trace-driven simulation relies on input data from an external source to supply recorded realizations of naturally occurring stochastic processes. Total reliance on such external data limits the applicability of a discrete-event simulation program, naturally inhibiting the user's ability to do "what if" studies. Given this limitation, a general discrete-event simulation objective is to develop methods for using a random number generator to convert a trace-driven discrete-event simulation program to a discrete-event simulation program that is not dependent on external data. This chapter provides several examples.

3.1.1 SINGLE-SERVER SERVICE NODE

Relative to the single-server service node model from Chapter 1, two stochastic assumptions are needed to free program `ssq1` from its reliance on external data. One assumption relates to the arrival times, the other assumption relates to the service times. We consider the service times first, using a $Uniform(a, b)$ random variate model.

Example 3.1.1 Suppose that time is measured in minutes in a single-server service node model and all that is known about the service time is that it is random with possible values between 1.0 and 2.0. That is, although we know the range of possible values, we are otherwise in such a state of ignorance about the stochastic behavior of this server that we are unwilling to say some service times (between 1.0 and 2.0) are more likely than others. In this case we have modeled service time as a $Uniform(1.0, 2.0)$ random variable. Accordingly, random variate service times, say s , can be generated via the assignment

```
s = Uniform(1.0, 2.0);
```

Exponential Random Variates

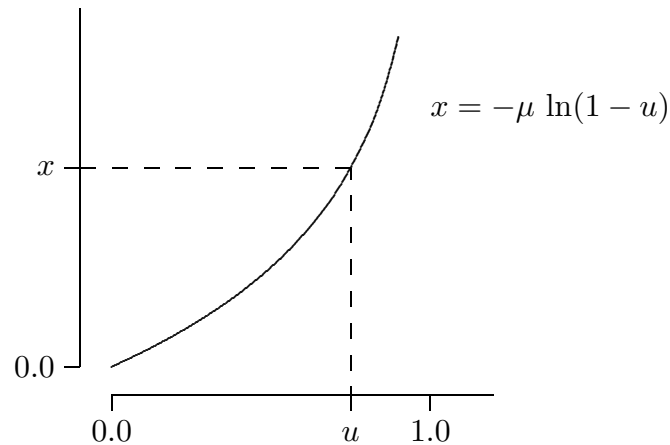
A $Uniform(a, b)$ random variable has the property that all values between a and b are equally likely. In most applications this is an unrealistic assumption; instead, some values will be more likely than others. Specifically, there are many discrete-event simulation applications that require a continuous random variate, say x , that can take on any positive value but in such a way that small values of x are more likely than large values.

To generate such a random variate we need a *nonlinear* transformation that maps values of the random number u between 0.0 and 1.0 to values of x between 0.0 and ∞ and does so by "stretching" large values of u much more so than small values. Although a variety of such nonlinear transformation are possible, for example $x = u/(1 - u)$, perhaps the most common is $x = -\mu \ln(1 - u)$ where $\mu > 0$ is a parameter that controls the rate of stretching and $\ln(\cdot)$ is the natural logarithm (base e).^{*} As explained in Chapter 7, this transformation generates what is known as an $Exponential(\mu)$ random variate.

^{*} The function $\log(x)$ in the ANSI C library `<math.h>` represents the mathematical natural log function $\ln(x)$. In the equation $x = -\mu \ln(1 - u)$ do not confuse the positive, real-valued parameter μ with the $Uniform(0, 1)$ random variate u .

It is often appropriate to generate interarrivals as *Exponential*(μ) random variates, as we will do later in Example 3.1.2. In some important theoretical situations, service times are also modeled this way (see Section 8.5). The geometry associated with the *Exponential*(μ) random variate transformation $x = -\mu \ln(1 - u)$ is illustrated in Figure 3.1.1.

Figure 3.1.1.
Exponential variate
generation geometry.



By inspection we see that the transformation is monotone increasing and, for any value of the parameter $\mu > 0$, the interval $0 < u < 1$ is mapped one-to-one and onto the interval $0 < x < \infty$. That is,

$$\begin{aligned} 0 < u < 1 &\iff 0 < (1 - u) < 1 \\ &\iff -\infty < \ln(1 - u) < 0 \\ &\iff 0 < -\mu \ln(1 - u) < \infty \\ &\iff 0 < x < \infty. \end{aligned}$$

Definition 3.1.1 This ANSI C function generates an *Exponential*(μ) random variate*

```
double Exponential(double  $\mu$ )                               /* use  $\mu > 0.0$  */
{
    return (- $\mu$  * log(1.0 - Random()));
}
```

The statistical significance of the parameter μ is that if repeated calls to the function *Exponential*(μ) are used to generate a random variate sample x_1, x_2, \dots, x_n then, in the limit as $n \rightarrow \infty$, the sample mean (average) of this sample will converge to μ . In the same sense, repeated calls to the function *Uniform*(a, b) will produce a sample whose mean converges to $(a + b)/2$ and repeated calls to the function *Equilikely*(a, b) will also produce a sample whose mean converges to $(a + b)/2$.

* The ANSI C standard says that `log(0.0)` may produce “a range error”. This possibility is naturally avoided in the function `Exponential` because the largest possible value of `Random` is less than 1.0. See Exercise 3.1.3.

Example 3.1.2 In a single-server service node simulation, to generate a sequence of random variate *arrival* times $a_1, a_2, a_3, \dots, a_n$ with an average interarrival time that will converge to μ as $n \rightarrow \infty$ it is common to generate *Exponential*(μ) *interarrival* times (see Definition 1.2.3) and then (with $a_0 = 0$) create the arrival times by the assignment

$$a_i = a_{i-1} + \text{Exponential}(\mu); \quad i = 1, 2, 3, \dots, n$$

As discussed in Chapter 7, this use of an *Exponential*(μ) random variate corresponds naturally to the idea of jobs arriving *at random* with an arrival rate that will converge to $1/\mu$ as $n \rightarrow \infty$. Similarly, as in Example 3.1.1, to generate a random variate sequence of service times $s_1, s_2, s_3, \dots, s_n$ equally likely to lie anywhere between a and b (with $0 \leq a < b$) and with an average service time that converges to $(a + b)/2$, the assignment

$$s_i = \text{Uniform}(a, b); \quad i = 1, 2, 3, \dots, n$$

can be used. The average service time $(a + b)/2$ corresponds to a service rate of $2/(a + b)$.

Program `ssq2`

Program `ssq2` is based on the two stochastic modeling assumptions in Example 3.1.2.* Program `ssq2` is an extension of program `ssq1` in that the arrival times and service times are generated randomly (rather than relying on a trace-driven input) and a complete set of first-order statistics $\bar{r}, \bar{w}, \bar{d}, \bar{s}, \bar{l}, \bar{q}, \bar{x}$ is generated. Note that each time the function `GetArrival()` is called the static variable `arrival`, which represents an *arrival* time, is incremented by a call to the function `Exponential(2.0)`, which generates an *interarrival* time.

Because program `ssq2` generates stochastic data as needed there is essentially no restriction on the number of jobs that can be processed. Therefore, the program can be used to study the *steady-state* behavior of a single-server service node. That is, by experimenting with an increasing number of jobs processed, one can investigate whether or not the service node statistics will converge to constant values, *independent* of the choice of the `rng` initial seed and the initial state of the service node. Steady-state behavior — can it be achieved and, if so, how many jobs does it take to do so — is an important issue that will be explored briefly in this chapter and in more detail in Chapter 8.

Program `ssq2` can also be used to study the *transient* behavior of a single-server service node. The idea in this case is to fix the number of jobs processed at some finite value and run (replicate) the program repeatedly with the initial state of the service node fixed, changing *only* the `rng` initial seed from run to run. In this case replication will produce a natural variation in the service node statistics consistent with the fact that for a fixed number of jobs, the service node statistics are *not* independent of the initial seed or the initial state of the service node. Transient behavior, and its relation to steady-state behavior, will be considered in Chapter 8.

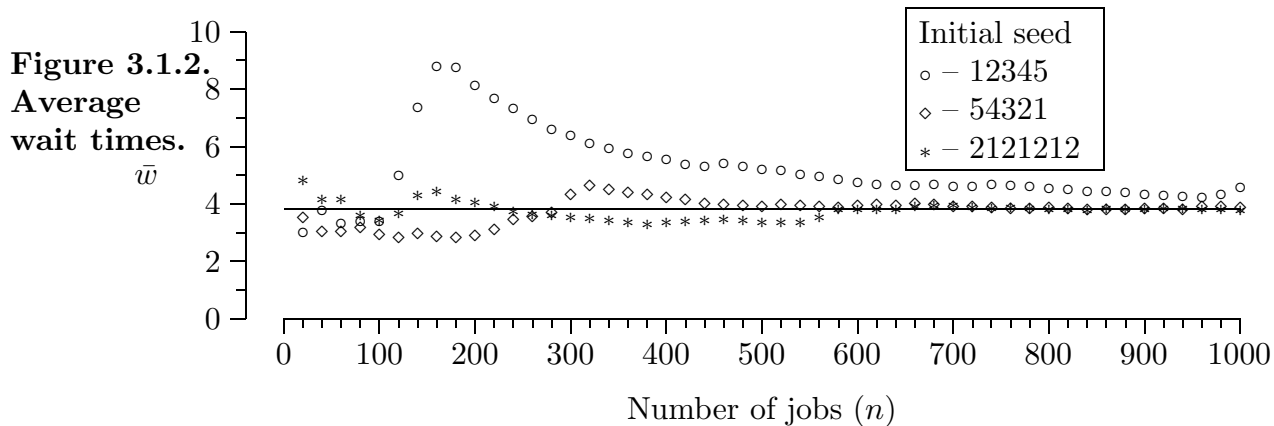
* In the jargon of queuing theory (see Section 8.5), program `ssq2` simulates what is known as an $M/G/1$ queue (see Kleinrock, 1975, 1976, or Gross and Harris, 1985).

Steady-State Statistics

Example 3.1.3 If the *Exponential*(μ) interarrival time parameter is set to $\mu = 2.0$ so that the steady-state arrival rate is $1/\mu = 0.5$ and if the *Uniform*(a, b) service time parameters are set to $a = 1.0$ and $b = 2.0$ respectively so that the steady-state service rate is $2/(a + b) \cong 0.67$, then to *d.dd* precision the theoretical steady-state statistics generated from an analytic model (exact, not estimates by simulation) program `ssq2` will produce are (see Gross and Harris, 1985)

\bar{r}	\bar{w}	\bar{d}	\bar{s}	\bar{l}	\bar{q}	\bar{x}
2.00	3.83	2.33	1.50	1.92	1.17	0.75

Therefore, although the server is only busy 75% of the time ($\bar{x} = 0.75$), “on average” there are approximately two jobs ($\bar{l} = 23/12 \cong 1.92$) in the service node and a job can expect to spend more time ($\bar{d} = 23/6 - 3/2 \cong 2.33$ time units) in the queue than in service ($\bar{s} = 1.50$ time units). As illustrated in Figure 3.1.2 for the average wait, the number of jobs that must be pushed through the service node to achieve these steady-state statistics is large. To produce this figure, program `ssq2` was modified to print the accumulated average wait every 20 jobs. The results are presented for three choices of the `rng` initial seed.* The solid, horizontal line at height $23/6 \cong 3.83$ represents the steady-state value of \bar{w} .



In Example 3.1.3 convergence of \bar{w} to the steady-state value 3.83 is slow, erratic, and very dependent on the random variate sequence of stochastic arrival and service times, as manifested by the choice of initial seed. Note, for example, the dramatic rise in average wait beginning at about job 100 associated with the `rng` initial seed 12345. You are encouraged to add diagnostic printing and additional statistics gathering to program `ssq2` to better understand what combination of chance events occurred to produce this rise.

The *Uniform*(a, b) service time model in Example 3.1.3 may not be realistic. Service times seldom “cut off” beyond a minimum and maximum value. More detail will be given in subsequent chapters on how to select realistic distributions for input models.

* There is nothing special about these three initial seeds. Do not fall into the common trap of thinking that some `rng` initial seeds are necessarily better than others.

Example 3.1.3 shows that the stochastic character of the arrival times and service times, as manifested by the choice of `rng` initial seed, has a significant effect on the transition-to-steady-state behavior of a single-server service node. This example also illustrates the use of the library `rng` to conduct *controlled* “what if” experiments. Studies like this are the stuff of discrete-event simulation. Additional examples of this kind of experimentation, and the selection of values of μ , a , and b , are presented in later chapters.

Geometric Random Variates

As discussed in Chapter 2, an *Equilikely*(a, b) random variate is the discrete analog of a continuous *Uniform*(a, b) random variate. Consistent with this characterization, one way to generate an *Equilikely*(a, b) random variate is to generate a *Uniform*($a, b + 1$) random variate instead (note that the upper limit is $b + 1$, not b) and then convert (cast) the resulting floating-point result to an integer. That is, if a and b are integers with $a < b$ and if x is a *Uniform*($a, b + 1$) random variate then $\lfloor x \rfloor$ is an *Equilikely*(a, b) random variate.

Given the analogy between *Uniform*(a, b) and *Equilikely*(a, b) random variates, it is reasonable to expect that there is a discrete analog to a continuous *Exponential*(μ) random variate; and there is. Specifically, if x is an *Exponential*(μ) random variate, let y be the *discrete* random variate defined by $y = \lfloor x \rfloor$. For a better understanding of this discrete random variate, let $p = \Pr(y \neq 0)$ denote the probability that y is not zero. Since x is generated as $x = -\mu \ln(1 - u)$ with u a *Uniform*($0, 1$) random variate, $y = \lfloor x \rfloor$ will not be zero if and only if $x \geq 1$. Equivalently

$$\begin{aligned} x \geq 1 &\iff -\mu \ln(1 - u) \geq 1 \\ &\iff \ln(1 - u) \leq -1/\mu \\ &\iff 1 - u \leq \exp(-1/\mu) \end{aligned}$$

where $\exp(-1/\mu)$ is $e^{-1/\mu}$, and so $y \neq 0$ if and only if $1 - u \leq \exp(-1/\mu)$. Like u , $1 - u$ is also a *Uniform*($0, 1$) random variate. Moreover, for any $0 < p < 1$, the condition $1 - u \leq p$ is true with probability p (see Section 7.1). Therefore, $p = \Pr(y \neq 0) = \exp(-1/\mu)$.

If x is an *Exponential*(μ) random variate and if $y = \lfloor x \rfloor$ with $p = \exp(-1/\mu)$, then it is conventional to call y a *Geometric*(p) random variable (see Chapter 6). Moreover, it is conventional to use p rather than $\mu = -1/\ln(p)$ to define y directly by the equation

$$y = \lfloor \ln(1 - u) / \ln(p) \rfloor.$$

Definition 3.1.2 This ANSI C function generates a *Geometric*(p) random variate*

```
long Geometric(double p)                /* use 0.0 < p < 1.0 */
{
    return ((long) (log(1.0 - Random()) / log(p)));
}
```

* Note that $\log(0.0)$ is avoided in this function because the largest possible value returned by `Random` is less than 1.0.

In addition to its significance as $\Pr(y \neq 0)$, the parameter p is also related to the mean of a *Geometric*(p) sample. Specifically, if repeated calls to the function `Geometric(p)` are used to generate a random variate sample y_1, y_2, \dots, y_n then, in the limit as $n \rightarrow \infty$, the mean of this sample will converge to $p/(1-p)$. Note that if p is close to 0.0 then the mean will be close to 0.0. At the other extreme, if p is close to 1.0 then the mean will be large.

In the following example, a *Geometric*(p) random variate is used as part of a *composite* service time model. In this example the parameter p has been adjusted to make the average service time match that of the *Uniform*(1.0, 2.0) server in program `ssq2`.

Example 3.1.4 Usually one has sufficient information to argue that a *Uniform*(a, b) random variate service time model is not appropriate; instead, a more sophisticated model is justified. Consider a hypothetical server that, as in program `ssq2`, processes a stream of jobs arriving, at random, with a steady-state arrival rate of 0.5 jobs per minute. The service requirement associated with each arriving job has two stochastic components:

- the *number* of service tasks is one plus a *Geometric*(0.9) random variate;
- the *time* (in minutes) per task is, independently for each task, a *Uniform*(0.1, 0.2) random variate.

In this case, program `ssq2` would need to be modified by including the function `Geometric` from Definition 3.1.2 and changing the function `GetService` to the following.

```
double GetService(void)
{
    long    k;
    double sum    = 0.0;
    long    tasks = 1 + Geometric(0.9);

    for (k = 0; k < tasks; k++)
        sum += Uniform(0.1, 0.2);
    return (sum);
}
```

With this modification, the population steady-state statistics from the analytic model to *d.dd* precision that will be produced are:

\bar{r}	\bar{w}	\bar{d}	\bar{s}	\bar{l}	\bar{q}	\bar{x}
2.00	5.77	4.27	1.50	2.89	2.14	0.75

Program `ssq2` will produce results that converge to these values for a very long run. When compared with the steady-state results in Example 3.1.3, note that although the arrival rate $1/\bar{r} = 0.50$, the service rate $1/\bar{s} = 0.67$, and the utilization $\bar{x} = 0.75$ are the same, the other four statistics are significantly larger than in Example 3.1.3. This difference illustrates the sensitivity of the performance measures to the service time distribution. This highlights the importance of using an accurate service time model. See Exercise 3.1.5.

3.1.2 SIMPLE INVENTORY SYSTEM

Example 3.1.5 In a simple inventory system simulation, to generate a random variate sequence of demands d_1, d_2, d_3, \dots equally likely to have any integer value between a and b inclusive and with an average of $(a + b)/2$ use

$$d_i = \text{Equilikely}(a, b) \quad i = 1, 2, 3, \dots$$

Recognize, however, that the previous discussion about $Uniform(a, b)$ service times as an unrealistic model also applies here in the discrete case. That is, in this application the modeling assumption that all the demands between a and b are equally likely is probably unrealistic; some demands should be more likely than others. As an alternative model we could consider generating the demands as $Geometric(p)$ random variates. In this particular case, however, a $Geometric(p)$ demand model is probably not very realistic either. In Chapter 6 we will consider alternative models that are more appropriate.

Program sis2

Program `sis2` is based on the $Equilikely(a, b)$ stochastic modeling assumption in Example 3.1.5. This program is an extension of program `sis1` in that the demands are generated randomly (rather than relying on a trace-driven input). Consequently, in a manner analogous to that illustrated in Example 3.1.3, program `sis2` can be used to study the transition-to-steady-state behavior of a simple inventory system.

Example 3.1.6 If the $Equilikely(a, b)$ demand parameters are set to $(a, b) = (10, 50)$ so that the average demand per time interval is $(a + b)/2 = 30$, then with $(s, S) = (20, 80)$ the (approximate) steady-state statistics program `sis2` will produce are

$$\begin{array}{ccccc} \bar{d} & \bar{o} & \bar{u} & \bar{l}^+ & \bar{l}^- \\ 30.00 & 30.00 & 0.39 & 42.86 & 0.26 \end{array}$$

As illustrated in Figure 3.1.3 (using the same three `rng` initial seeds used in Example 3.1.3) for the average inventory level $\bar{l} = \bar{l}^+ - \bar{l}^-$, at least several hundred time intervals must be simulated to approximate these steady-state statistics. To produce Figure 3.1.3, program `sis2` was modified to print the accumulated value of \bar{l} every 5 time intervals.

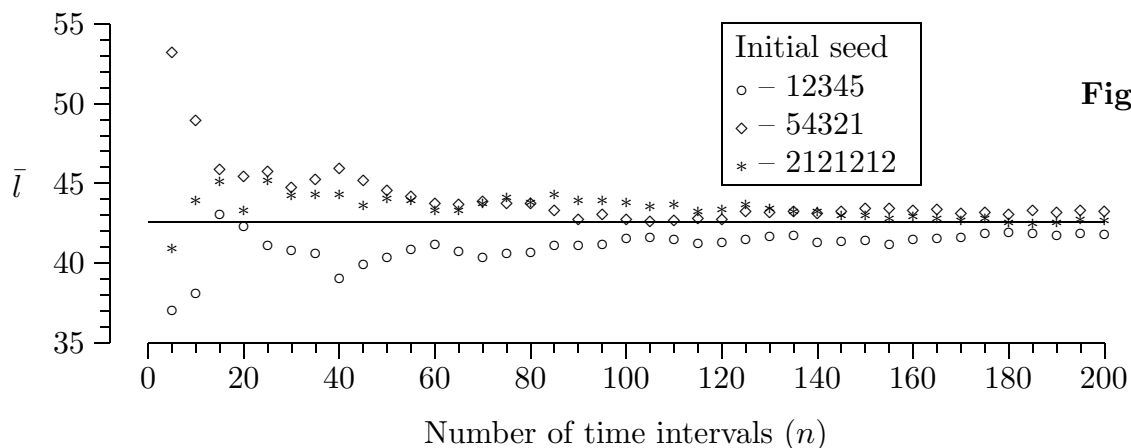


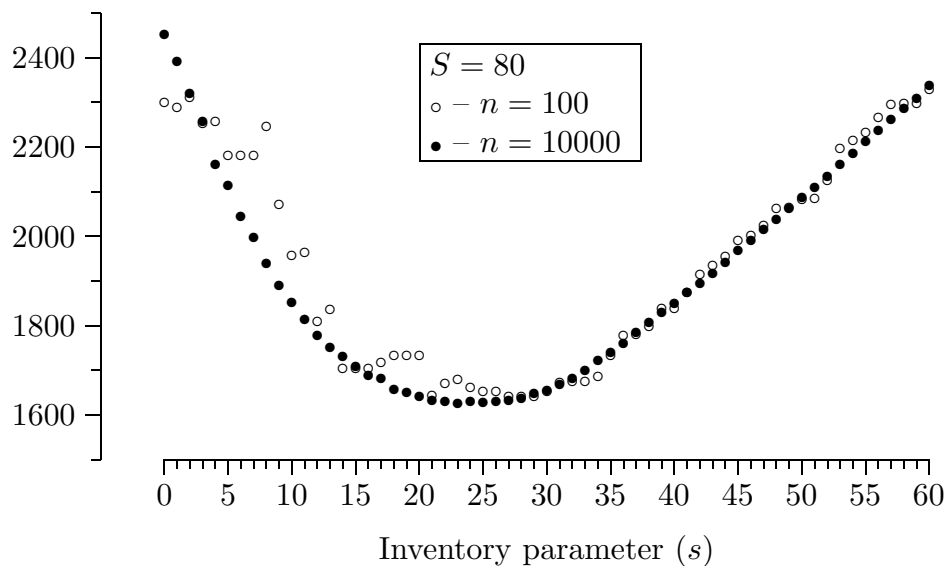
Figure 3.1.3.
Average
inventory
level.

Optimal Steady-State System Performance

Example 3.1.7 As an extension of Example 1.3.7, a modified version of program `sis2` was used to simulate $n = 100$ weeks (about two years) of automobile dealership operation with $S = 80$ and values of s varied from 0 to 60. The cost parameters from Example 1.3.5 were used to determine the variable part of the dealership's average weekly cost of operation. That is, as discussed in Section 1.3, the (large) part of the average weekly cost that is proportional to the average weekly order \bar{o} , and therefore is independent of (s, S) , was ignored so that only the dependent cost was computed. The results are presented with \circ 's representing the $n = 100$ averages. As illustrated in Figure 3.1.4, there is a relatively well-defined minimum with an optimal value of s somewhere between 20 and 30.

Figure 3.1.4.
Dependent
cost for (s, S)
inventory
system.

dependent
cost, \$



The same initial seed (12345) was used for each value of s . Fixing the initial seed guarantees that *exactly* the same sequence of demands will be processed; therefore, any changes in the resulting system statistics are due to changes in s only. As in Example 3.1.6, the demands were generated as *Equilikely*(10, 50) random variates. To compute steady-state statistics, all the modeling assumptions upon which this simulation is based would have to remain valid for many years. For that reason, steady-state statistics are not very meaningful in this example. Indeed, it is questionable to assume that the demand distribution, cost parameters, and inventory policy will remain constant for even two years. Steady-state performance statistics are fashionable, however, and represent an interesting limiting case. For that reason averages based on $n = 10000$ weeks of operation (approximately 192 years) are presented as \bullet 's for comparison. These estimated steady-state averages have the attractive feature that they vary smoothly with s and, because of that, the optimum steady-state value of s is relatively well defined. In a world that (on average) never changes, if $S = 80$ the auto dealer should use an inventory threshold of $s = 23$. This example illustrates the more general problem of optimizing a function that is not known with certainty known as *stochastic optimization*.

3.1.3 STATISTICAL CONSIDERATIONS

The statistical analysis of simulation-generated data is discussed in Chapters 4 and 8. In anticipation of that discussion, we note that Example 3.1.7 illustrates two important ideas: *variance reduction* and *robust estimation*. We consider variance reduction first, particularly as it relates to transient (small n) output statistics.

Variance Reduction

Because the transient statistical output produced by any discrete-event simulation program depends on the sequence of random variates generated as the program executes, the output will always have some inherent uncertainty. That is, as illustrated in Examples 3.1.3 and 3.1.6, the transient statistical output depends on the value of the `rng` initial seed, particularly if the number of jobs or the number of simulated time intervals is small. Therefore, if a variety of initial seeds are used (a policy we certainly advocate) there will be a natural *variance* in any computed transient statistic.

The statistical tools used to quantify this variance/uncertainty will be developed in Chapter 8. For now, the key point to be made is that using *common random numbers*, as in Example 3.1.7 when $n = 100$, is an intuitive approach to reducing the variance in computed system statistics. This example of variance reduction (using common random numbers) is consistent with the time-honored approach to experimental scientific “what if” studies where, if possible, all variables except one are fixed. Fixing the initial seed at 12345 in Example 3.1.7 isolates the variability of the performance measure (dependent cost). This technique is more generally known as *blocking* in statistics.

Robust Estimation

The optimal (minimal cost) estimated steady-state threshold value $s = 23$ in Example 3.1.7 is a robust estimate because other values of s close to 23 yield essentially the same cost. Therefore, the impact of using one of these alternate values of s in place of $s = 23$ would be slight. In a more general sense, it is desirable for an optimal value to be robust to *all* the assumptions upon which the discrete-event simulation model is based. Relative to Example 3.1.7, this means that we would be interested in determining what happens to the optimal value of s when, for example S is varied about 80, or the average demand (per week) is varied about 30, or there are delivery lags. The estimate $s = 23$ is robust only if it remains close to the minimal cost operating policy level when all these assumptions are altered. In general, robust estimators are insensitive to model assumptions.

3.1.4 EXERCISES

Exercise 3.1.1 (a) Modify program `ssq2` to use *Exponential*(1.5) service times. (b) Process a relatively large number of jobs, say 100 000, and determine what changes this produces relative to the statistics in Example 3.1.3? (c) Explain (or conjecture) why some statistics change and others do not.

Exercise 3.1.2 (a) Relative to the steady-state statistics in Example 3.1.3 and the statistical equations in Section 1.2, list all of the consistency checks that should be applicable. (b) Verify that all of these consistency checks are valid.

Exercise 3.1.3 (a) Given that the Lehmer random number generator used in the library `rng` has a modulus of $2^{31} - 1$, what are the largest and smallest possible numerical values (as a function of μ) that the function `Exponential(μ)` can return? (b) Comment on this relative to the theoretical expectation that an *Exponential(μ)* random variate can have an arbitrarily large value and a value arbitrarily close to zero.

Exercise 3.1.4 (a) Conduct a transition-to-steady-state study like that in Example 3.1.3 except for a service time model that is *Uniform(1.3, 2.3)*. Be specific about the number of jobs that seem to be required to produce steady-state statistics. (b) Comment.

Exercise 3.1.5 (a) Verify that the mean service time in Example 3.1.4 is 1.5. (b) Verify that the steady-state statistics in Example 3.1.4 seem to be correct. (c) Note that the arrival rate, service rate, and utilization are the same as those in Example 3.1.3, yet all the other statistics are larger than those in Example 3.1.3. Explain (or conjecture) why this is so. Be specific.

Exercise 3.1.6 (a) Modify program `sis2` to compute data like that in Example 3.1.7. Use the functions `PutSeed` and `GetSeed` from the library `rng` in such a way that *one* initial seed is supplied by the system clock, printed as part of the program's output and used automatically to generate the same demand sequence for all values of s . (b) For $s = 15, 16, \dots, 35$ create a figure (or table) similar to the one in Example 3.1.7. (c) Comment.

Exercise 3.1.7 (a) Relative to Example 3.1.5, if instead the random variate sequence of demands are generated as

$$d_i = \text{Equilikely}(5, 25) + \text{Equilikely}(5, 25) \quad i = 1, 2, 3, \dots$$

then, when compared with those in Example 3.1.6, demonstrate that some of the steady-state statistics will be the same and others will not. (b) Explain why this is so.

Exercise 3.1.8^a Modify program `sis2` to simulate the operation of a simple inventory system *with a delivery lag*. (a) Specifically, assume that if an order is placed at time $t = i - 1$ then the order will arrive at the later time $t = i - 1 + \delta_i$ where the delivery lag δ_i is a *Uniform(0, 1)* random variate, independent of the size of the order. (b) What are the equations for \bar{l}_i^+ and \bar{l}_i^- ? (c) Using the same parameter values as in Example 3.1.7, determine that value of s for which the average dependent cost is least. Compare this result with that obtained in Example 3.1.7. (d) It is important to have the *same* sequence of demands for all values of s , with and without a lag. Why? How did you accomplish this? (e) Discuss what you did to convince yourself that the modification is correct. (f) For both $n = 100$ and $n = 10000$ produce a table of dependent costs corresponding to $s = 10, 20, 30, 40, 50, 60$.