```
Single-server queueing system with fixed run length

Mean interarrival time          1.000 minutes

Mean service time               0.500 minutes

Length of the simulation      480.000 minutes


Average delay in queue          0.399 minutes

Average number in queue         0.394

Server utilization              0.464

Number of delays completed        475
```

FIGURE 1.37
Output report, queueing model with fixed run length.

1.4.8 Determining the Events and Variables

We defined an event in Sec. 1.3 as an instantaneous occurrence that may change the system state, and in the simple single-server queue of Sec. 1.4.1 it was not too hard to identify the events. However, the question sometimes arises, especially for complex systems, of how one determines the number and definition of events in general for a model. It may also be difficult to specify the state variables needed to keep the simulation running in the correct event sequence and to obtain the desired output measures. There is no completely general way to answer these questions, and different people may come up with different ways of representing a model in terms of events and variables, all of which may be correct. But there are some principles and techniques to help simplify the model's structure and to avoid logical errors.

Schruben (1983b) presented an *event-graph* method, which was subsequently refined and extended by Sargent (1988) and Som and Sargent (1989). In this approach proposed events, each represented by a *node*, are connected by *directed arcs* (arrows) depicting how events may be scheduled from other events and from themselves. For example, in the queueing simulation of Sec. 1.4.3, the arrival event schedules another future occurrence of itself and (possibly) a departure event, and the departure event may schedule another future occurrence of itself; in addition, the arrival event must be initially scheduled in order to get the simulation going. Event graphs connect the proposed set of events (nodes) by arcs indicating the type of event scheduling that can occur. In Fig. 1.38 we show the event graph for our

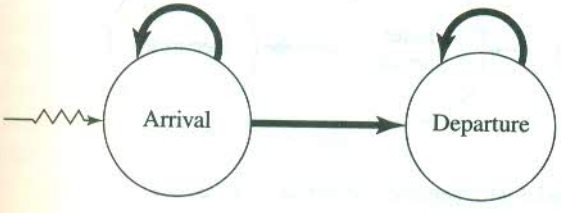

FIGURE 1.38
Event graph, queueing model.

single-server queueing system, where the heavy, smooth arrows indicate that an event at the end of the arrow *may* be scheduled from the event at the beginning of the arrow in a (possibly) *nonzero* amount of time, and the thin jagged arrow indicates that the event at its end is scheduled initially. Thus, the arrival event reschedules itself and may schedule a departure (in the case of an arrival who finds the server idle), and the departure event may reschedule itself (if a departure leaves behind someone else in queue).

For this model, it could be asked why we did not explicitly account for the act of a customer's entering service (either from the queue or upon arrival) as a separate event. This certainly can happen, and it could cause the state to change (i.e., the queue length to fall by 1). In fact, this could have been put in as a separate event without making the simulation incorrect, and would give rise to the event diagram in Fig. 1.39. The two thin smooth arrows each represent an event at the beginning of an arrow potentially scheduling an event at the end of the arrow without any intervening time, i.e., immediately; in this case the straight thin smooth arrow refers to a customer who arrives to an empty system and whose "enter-service" event is thus scheduled to occur immediately, and the curved thin smooth arrow represents a customer departing with a queue left behind, and so the first customer in the queue would be scheduled to enter service immediately. The number of events has now increased by 1, and so we have a somewhat more complicated representation of our model. One of the uses of event graphs is to simplify a simulation's event structure by eliminating unnecessary events. There are several "rules" that allow for simplification, and one of them is that if an event node has incoming arcs that are all thin and smooth (i.e., the only way this event is scheduled is by other events and without any intervening time), then this event can be eliminated from the model and its action built into the events that schedule it in zero time. Here, the "enter-service" event could be eliminated, and its action put partly into the arrival event (when a customer arrives to an idle server and begins service immediately) and partly into the departure event (when a customer finishes service and there is a queue from which the next customer is taken to enter service); this takes us back to the simpler event graph in Fig. 1.38. Basically, "events" that can happen only in conjunction with other events do not need to be in the model. Reducing the number of events not only simplifies model conceptualization, but may also speed its execution. Care must be taken, however, when "collapsing" events in this way to handle priorities and time ties appropriately.

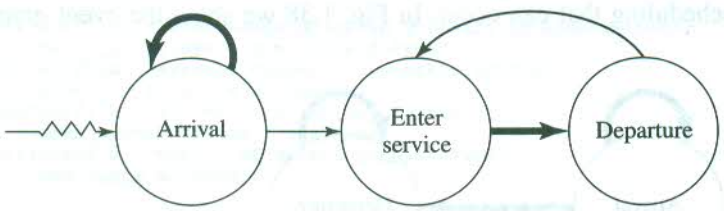

FIGURE 1.39
Event graph, queueing model with separate "enter-service" event.

Another rule has to do with initialization. The event graph is decomposed into *strongly connected* components, within each of which it is possible to "travel" from every node to every other node by following the arcs in their indicated directions. The graph in Fig. 1.38 decomposes into two strongly connected components (with a single node in each), and that in Fig. 1.39 has two strongly connected components (one of which is the arrival node by itself, and the other of which consists of the enter-service and departure nodes). The initialization rule states that in any strongly connected component of nodes that has no incoming arcs from other event nodes outside the component, there must be at least one node that is initially scheduled; if this rule were violated, it would never be possible to execute any of the events in the component. In Figs. 1.38 and 1.39, the arrival node is such a strongly connected component since it has no incoming arcs from other nodes, and so it must be initialized. Figure 1.40 shows the event graph for the queueing model of Sec. 1.4.7 with the fixed run length, for which we introduced the dummy "end-simulation" event. Note that this event is itself a strongly connected component without any arcs coming in, and so it must be initialized; i.e., the end of the simulation is scheduled as part of the initialization. Failure to do so would result in erroneous termination of the simulation.

We have presented only a partial and simplified account of the event-graph technique. There are several other features, including event-canceling relations, ways to combine similar events into one, refining the event-scheduling arcs to include conditional scheduling, and incorporating the state variables needed; see the original paper by Schruben (1983b). Sargent (1988) and Som and Sargent (1989) extend and refine the technique, giving comprehensive illustrations involving a flexible manufacturing system and computer network models. Event graphs can also be used to test whether two apparently different models might in fact be equivalent [Yücesan and Schruben (1992)], as well as to forecast how computationally intensive a model will be when it is executed [Yücesan and Schruben (1998)]. Schruben (1995) provides a software package, SIGMA, that allows on-screen building of an event-graph representation of a simulation model, and then generates code and runs the model. A general event-graph review and tutorial are given by Buss (1996).

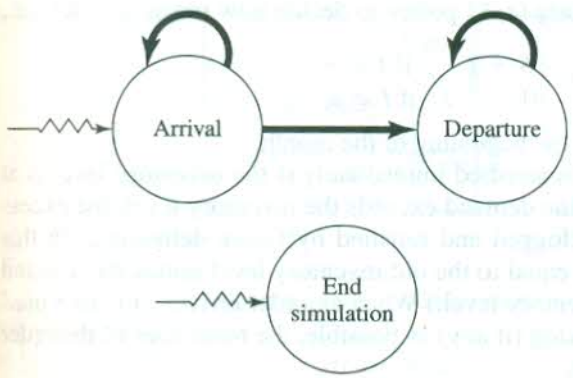

FIGURE 1.40
Event graph, queueing model with fixed run length.

In modeling a system, the event-graph technique can be used to simplify the structure and to detect certain kinds of errors, and is especially useful in complex models involving a large number of interrelated events. Other considerations should also be kept in mind, such as continually asking why a particular state variable is needed; see Prob. 1.4.

1.5 SIMULATION OF AN INVENTORY SYSTEM

We shall now see how simulation can be used to compare alternative ordering policies for an inventory system. Many of the elements of our model are representative of those found in actual inventory systems.

1.5.1 Problem Statement

A company that sells a single product would like to decide how many items it should have in inventory for each of the next n months (n is a fixed input parameter). The times between demands are IID exponential random variables with a mean of 0.1 month. The sizes of the demands, D, are IID random variables (independent of when the demands occur), with

$$D = \begin{cases} 1 & \text{w.p. } \frac{1}{6} \\ 2 & \text{w.p. } \frac{1}{3} \\ 3 & \text{w.p. } \frac{1}{3} \\ 4 & \text{w.p. } \frac{1}{6} \end{cases}$$

where w.p. is read "with probability."

At the beginning of each month, the company reviews the inventory level and decides how many items to order from its supplier. If the company orders Z items, it incurs a cost of $K + iZ$, where $K = \$32$ is the *setup cost* and $i = \$3$ is the *incremental cost* per item ordered. (If $Z = 0$, no cost is incurred.) When an order is placed, the time required for it to arrive (called the *delivery lag* or *lead time*) is a random variable that is distributed uniformly between 0.5 and 1 month.

The company uses a stationary (s, S) policy to decide how much to order, i.e.,

$$Z = \begin{cases} S - I & \text{if } I < s \\ 0 & \text{if } I \geq s \end{cases}$$

where I is the inventory level at the beginning of the month.

When a demand occurs, it is satisfied immediately if the inventory level is at least as large as the demand. If the demand exceeds the inventory level, the excess of demand over supply is backlogged and satisfied by future deliveries. (In this case, the new inventory level is equal to the old inventory level minus the demand size, resulting in a negative inventory level.) When an order arrives, it is first used to eliminate as much of the backlog (if any) as possible; the remainder of the order (if any) is added to the inventory.

So far, we have discussed only one type of cost incurred by the inventory system, the ordering cost. However, most real inventory systems also have two additional types of costs, *holding* and *shortage* costs, which we discuss after introducing some additional notation. Let $I(t)$ be the inventory level at time t [note that $I(t)$ could be positive, negative, or zero]; let $I^+(t) = \max\{I(t), 0\}$ be the number of items physically on hand in the inventory at time t [note that $I^+(t) \geq 0$]; and let $I^-(t) = \max\{-I(t), 0\}$ be the backlog at time t [$I^-(t) \geq 0$ as well]. A possible realization of $I(t)$, $I^+(t)$, and $I^-(t)$ is shown in Fig. 1.41. The time points at which $I(t)$ decreases are the ones at which demands occur.

For our model, we shall assume that the company incurs a holding cost of $h = \$1$ per item per month held in (positive) inventory. The holding cost includes such costs as warehouse rental, insurance, taxes, and maintenance, as well as the opportunity cost of having capital tied up in inventory rather than invested elsewhere. We have ignored in our formulation the fact that some holding costs are still incurred when $I^+(t) = 0$. However, since our goal is to *compare* ordering policies, ignoring this factor, which after all is independent of the policy used, will not affect our assessment of which policy is best. Now, since $I^+(t)$ is the number of items held in inventory at time t, the time-average (per month) number of items held in inventory for the n-month period is

$$\bar{I}^+ = \frac{\int_0^n I^+(t)\, dt}{n}$$

which is akin to the definition of the time-average number of customers in queue given in Sec. 1.4.1. Thus, the average holding cost per month is $h\bar{I}^+$.

Similarly, suppose that the company incurs a backlog cost of $\pi = \$5$ per item per month in backlog; this accounts for the cost of extra record keeping when a backlog exists, as well as loss of customers' goodwill. The time-average number of

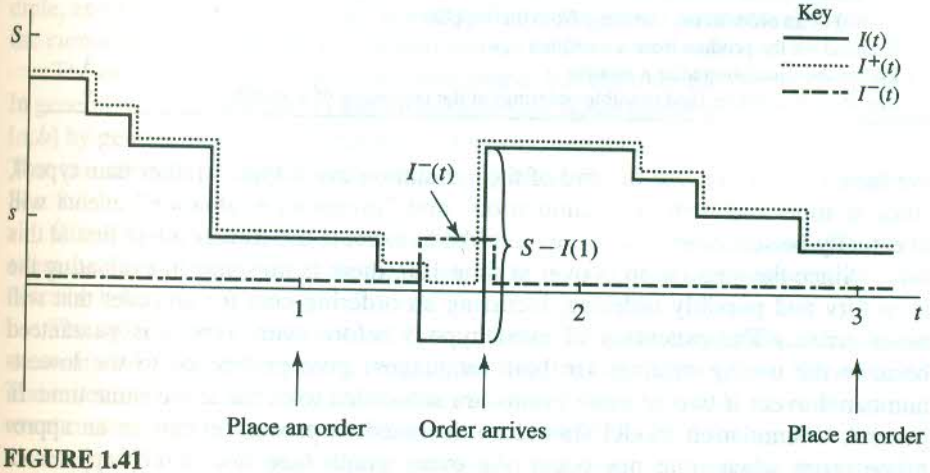

FIGURE 1.41
A realization of $I(t)$, $I^+(t)$, and $I^-(t)$ over time.

items in backlog is

$$\bar{I}^- = \frac{\int_0^n I^-(t)\,dt}{n}$$

so the average backlog cost per month is $\pi\bar{I}^-$.

Assume that the initial inventory level is $I(0) = 60$ and that no order is outstanding. We simulate the inventory system for $n = 120$ months and use the average total cost per month (which is the sum of the average ordering cost per month, the average holding cost per month, and the average shortage cost per month) to compare the following nine inventory policies:

s	20	20	20	20	40	40	40	60	60
S	40	60	80	100	60	80	100	80	100

We do not address here the issue of how these particular policies were chosen for consideration; statistical techniques for making such a determination are discussed in Chap. 12.

Note that the state variables for a simulation model of this inventory system are the inventory level $I(t)$, the amount of an outstanding order from the company to the supplier, and the time of the last event [which is needed to compute the areas under the $I^+(t)$ and $I^-(t)$ functions].

1.5.2 Program Organization and Logic

Our model of the inventory system uses the following types of events:

Event description	Event type
Arrival of an order to the company from the supplier	1
Demand for the product from a customer	2
End of the simulation after n months	3
Inventory evaluation (and possible ordering) at the beginning of a month	4

We have chosen to make the end of the simulation event type 3 rather than type 4, since at time 120 both "end-simulation" and "inventory-evaluation" events will eventually be scheduled and we would like to execute the former event first at this time. (Since the simulation is over at time 120, there is no sense in evaluating the inventory and possibly ordering, incurring an ordering cost for an order that will never arrive.) The execution of event type 3 before event type 4 is guaranteed because the timing routines (in both languages) give preference to the lowest-numbered event if two or more events are scheduled to occur at the same time. In general, a simulation model should be designed to process events in an appropriate order when time ties occur. An event graph (see Sec. 1.4.8) appears in Fig. 1.42.

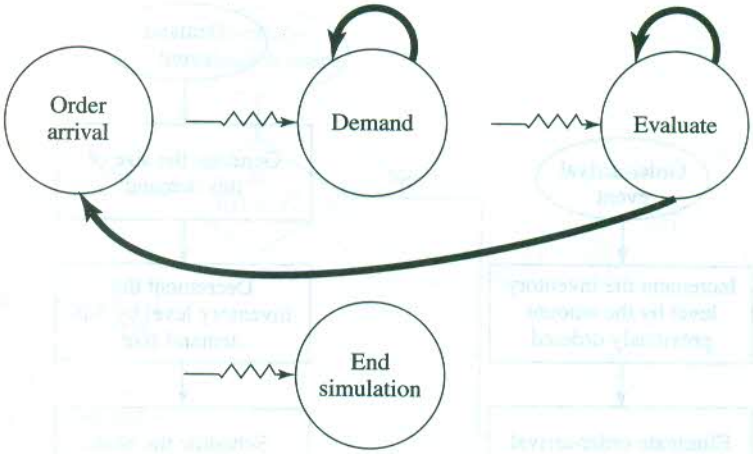

FIGURE 1.42
Event graph, inventory model.

There are three types of random variates needed to simulate this system. The interdemand times are distributed exponentially, so the same algorithm (and code) as developed in Sec. 1.4 can be used here. The demand-size random variate D must be discrete, as described above, and can be generated as follows. First divide the unit interval into the contiguous subintervals $C_1 = [0, \frac{1}{6})$, $C_2 = [\frac{1}{6}, \frac{1}{2})$, $C_3 = [\frac{1}{2}, \frac{5}{6})$, and $C_4 = [\frac{5}{6}, 1]$, and obtain a U(0, 1) random variate U from the random-number generator. If U falls in C_1, return $D = 1$; if U falls in C_2, return $D = 2$; and so on. Since the width of C_1 is $\frac{1}{6} - 0 = \frac{1}{6}$, and since U is uniformly distributed over [0, 1], the probability that U falls in C_1 (and thus that we return $D = 1$) is $\frac{1}{6}$; this agrees with the desired probability that $D = 1$. Similarly, we return $D = 2$ if U falls in C_2, having probability equal to the width of C_2, $\frac{1}{2} - \frac{1}{6} = \frac{1}{3}$, as desired; and so on for the other intervals. The subprograms to generate the demand sizes all use this principle, and take as input the cutoff points defining the above subintervals, which are the *cumulative* probabilities of the distribution of D.

The delivery lags are uniformly distributed, but not over the unit interval [0, 1]. In general, we can generate a random variate distributed uniformly over any interval $[a, b]$ by generating a U(0, 1) random number U, and then returning $a + U(b - a)$. That this method is correct seems intuitively clear, but will be formally justified in Sec. 8.3.1.

Of the four events, only three actually involve state changes (the end-simulation event being the exception). Since their logic is language-independent, we will describe it here.

The order-arrival event is flowcharted in Fig. 1.43, and must make the changes necessary when an order (which was previously placed) arrives from the supplier. The inventory level is increased by the amount of the order, and the order-arrival event must be eliminated from consideration. (See Prob. 1.12 for consideration of the issue of whether there could be more than one order outstanding at a time for this model with these parameters.)

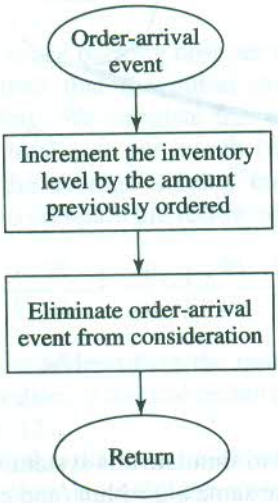

FIGURE 1.43
Flowchart for order-arrival routine, inventory model.

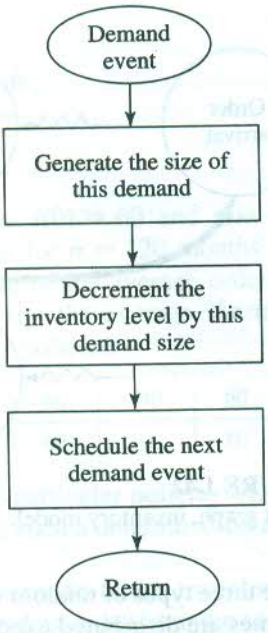

FIGURE 1.44
Flowchart for demand routine, inventory model.

A flowchart for the demand event is given in Fig. 1.44, and processes the changes necessary to represent a demand's occurrence. First, the demand size is generated, and the inventory is decremented by this amount. Finally, the time of the next demand is scheduled into the event list. Note that this is the place where the inventory level might become negative.

The inventory-evaluation event, which takes place at the beginning of each month, is flowcharted in Fig. 1.45. If the inventory level $I(t)$ at the time of the evaluation is at least s, then no order is placed, and nothing is done except to schedule the next evaluation into the event list. On the other hand, if $I(t) < s$, we want to place an order for $S - I(t)$ items. This is done by storing the amount of the order $[S - I(t)]$ until the order arrives, and scheduling its arrival time. In this case as well, we want to schedule the next inventory-evaluation event.

As in the single-server queueing model, it is convenient to write a separate nonevent routine to update the continuous-time statistical accumulators. For this model, however, doing so is slightly more complicated, so we give a flowchart for this activity in Fig. 1.46. The principal issue is whether we need to update the area under $I^-(t)$ or $I^+(t)$ (or neither). If the inventory level since the last event has been negative, then we have been in backlog, so the area under $I^-(t)$ only should be updated. On the other hand, if the inventory level has been positive, we need only update the area under $I^+(t)$. If the inventory level has been zero (a possibility), then

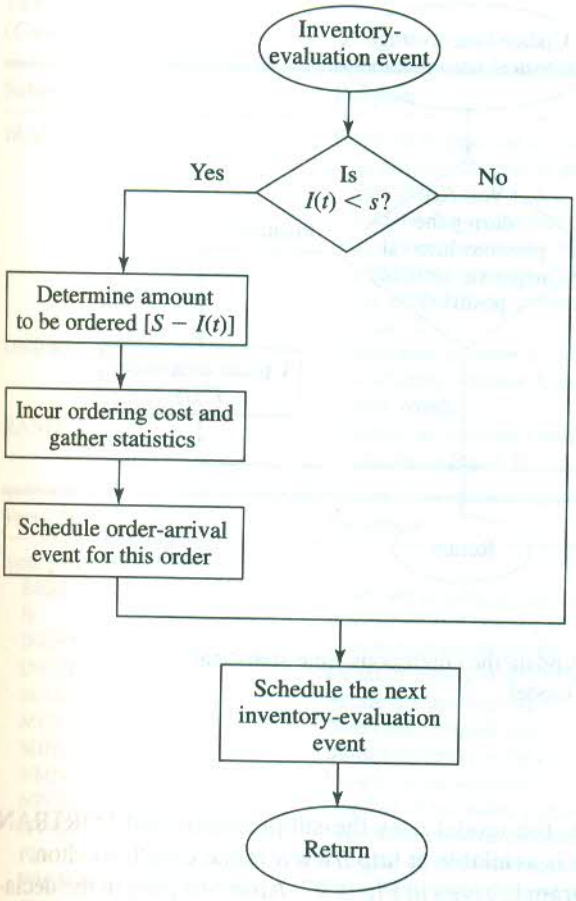

FIGURE 1.45
Flowchart for inventory-evaluation routine, inventory model.

neither update is needed. The code in both languages for this routine also brings the variable for the time of the last event up to the present time. This routine will be invoked from the main program just after returning from the timing routine, regardless of the event type or whether the inventory level is actually changing at this point. This provides a simple (if not the most computationally efficient) way of updating integrals for continuous-time statistics.

Sections 1.5.3 and 1.5.4, respectively, contain programs to simulate this model in FORTRAN and C. As in the single-server queueing model, only one of these sections should be read, according to language preference. Neither the timing nor exponential-variate-generation subprograms will be shown, as they are the same as for the single-server queueing model in Sec. 1.4 (except for the FORTRAN version of TIMING, where the declarations file "mm1.dcl" must be changed to "inv.dcl" in the INCLUDE statement). The reader should also note the considerable similarity between the main programs of the queueing and inventory models in a given language.

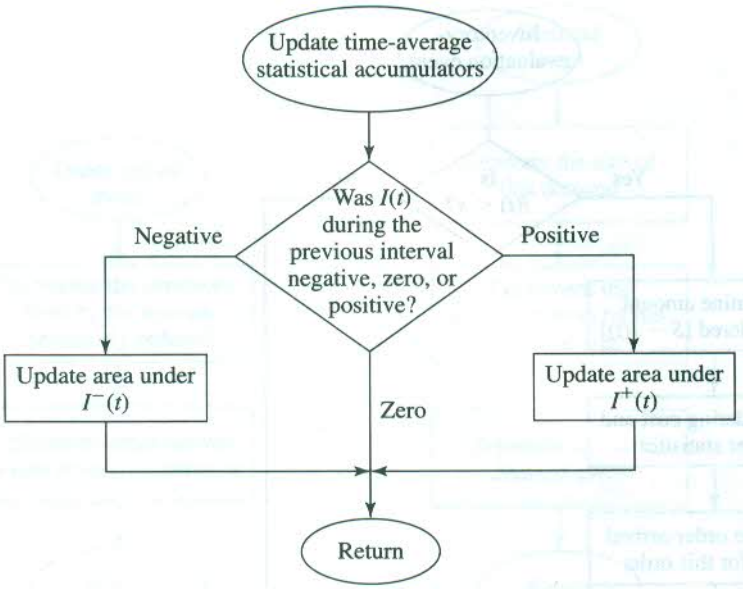

FIGURE 1.46
Flowchart for routine to update the continuous-time statistical accumulators, inventory model.

1.5.3 FORTRAN Program

In addition to a main program, the model uses the subprograms and FORTRAN variables in Table 1.2. All code is available at http://www.mhhe.com/lawkelton.

The code for the main program is given in Fig. 1.47. After bringing in the declarations file (shown in Fig. 1.48) and declaring the local variables I and NPOLCY to be INTEGER, the input and output files are opened, and the number of events, NEVNTS, is set to 4. The input parameters (except s and S) are then read in and

TABLE 1.2
Subroutines, functions, and FORTRAN variables for the inventory model

Subprogram	Purpose
INIT	Initialization routine
TIMING	Timing routine
ORDARV	Event routine to process type 1 events
DEMAND	Event routine to process type 2 events
REPORT	Event routine to process type 3 events (report generator)
EVALU8	Event routine to process type 4 events
UPTAVG	Subroutine to update areas under $I^+(t)$ and $I^-(t)$ functions just before each event occurrence
EXPON(RMEAN)	Function to generate an exponential random variate with mean RMEAN

(*Continued*)

moving forward at all times. Furthermore, with the time-warp mechanism, deadlocks are avoided.

Development and evaluation of distributed processing in simulation is currently an active area of research. It does seem clear, however, that how well (or even whether) a particular method will work may depend on the model's structure and parameters, as well as on the computing environment available. For example, if a model can be decomposed into submodels that are only weakly related (e.g., a network of queues in which customers only rarely move from one queue to another), then either of the model-based schemes discussed above for distributing the simulation may be expected to show more promise. For specific investigations into the efficacy of distributed simulation, see Lavenberg, Muntz, and Samadi (1983), Comfort (1984), and Heidelberger (1988); the last of these papers considers the impact of distributed simulation on statistical (rather than run-time) efficiency, with mixed results. How best to break up a simulation model for distribution across parallel processors, including the possibility of altering this breakup dynamically during the run, is investigated by Shanker, Kelton, and Padman (1993) and Shanker, Padman, and Kelton (2000).

1.6.2 Simulation across the Internet and Web-Based Simulation

With the rapid development of the Internet and the World Wide Web, a natural question arises as to how this gigantic (yet largely uncontrolled) network might be used to build, share, modify, distribute, and run simulations. Fishwick (1996, 1997) explores a wide range of issues in this regard, including client-server arrangements for increasing processing power, dissemination of simulation models and results, publication, education, and training. A general discussion of approaches to Web-based simulation, along with a specific operating example, is described by Lorenz et al. (1997). While it is difficult to predict precisely how the Internet and the Web might affect simulation, it seems clear that interest is high and that many people are exploring a wide variety of ways to use this technology in novel ways to support simulations.

1.7 STEPS IN A SOUND SIMULATION STUDY

Now that we have looked in some detail at the inner workings of a discrete-event simulation, we need to step back and realize that model programming is just part of the overall effort to design or analyze a complex system by simulation. Attention must be paid to a variety of other concerns such as statistical analysis of simulation output data and project management. Figure 1.68 shows the steps that will compose a typical, sound simulation study [see also Banks, Carson, and Nelson (1996), Law (1999), and Law and McComas (1990)]. The number beside the symbol representing each step refers to the more detailed description of that step below. Note that a simulation study is not a simple sequential process. As one proceeds with the study, it may be necessary to go back to a previous step.

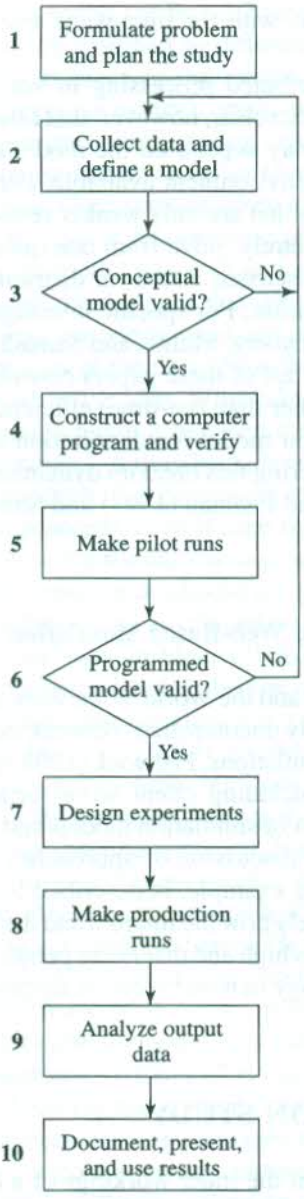

FIGURE 1.68
Steps in a simulation study.

1. *Formulate the problem and plan the study.*
   a. Problem of interest is stated by manager.
   b. One or more kickoff meetings for the study are conducted, with the project manager, the simulation analysts, and subject-matter experts (SMEs) in attendance. The following issues are discussed:
   - Overall objectives of the study
   - Specific questions to be answered by the study

- Performance measures that will be used to evaluate the efficacy of different system configurations
- Scope of the model
- System configurations to be modeled
- Software to be used (see Chap. 3)
- Time frame for the study and the required resources

2. *Collect data and define a model.*

   a. Collect information on the system layout and operating procedures.
      - No single person or document is sufficient.
      - Some people may have inaccurate information—make sure that true SMEs are identified.
      - Operating procedures may not be formalized.

   b. Collect data (if possible) to specify model parameters and input probability distributions (see Chap. 6).

   c. Delineate the above information and data in an "assumptions document," which is the *conceptual model* (see Sec. 5.4.3).

   d. Collect data (if possible) on the performance of the existing system (for validation purposes in Step 6).

   e. The level of model detail should depend on the following:
      - Project objectives
      - Performance measures
      - Data availability
      - Credibility concerns
      - Computer constraints
      - Opinions of SMEs
      - Time and money constraints

   f. There need not be a one-to-one correspondence between each element of the model and the corresponding element of the system.

   g. Interact with the manager (and other key project personnel) on a regular basis (see Sec. 5.4.2).

3. *Is the conceptual model valid?*

   a. Perform a structured walk-through of the conceptual model using the assumptions document before an audience of managers, analysts, and SMEs (see Sec. 5.4.3).
      - Helps ensure that the model's assumptions are correct and complete
      - Promotes ownership of the model
      - Takes place *before* programming begins to avoid significant reprogramming later

4. *Construct a computer program and verify.*

   a. Program the model in a programming language (e.g., C or FORTRAN) or in simulation software (e.g., Arena, AutoMod, Extend, ProModel, WITNESS).

Benefits of using a programming language are that one is often known, they have a low *purchase* cost, and they may result in a smaller model execution time. The use of simulation software (see Chap. 3), on the other hand, reduces programming time and results in a lower *project* cost.

b. Verify (debug) the simulation computer program (see Sec. 5.3).

5. *Make pilot runs.*

a. Make pilot runs for validation purposes in Step 6.

6. *Is the programmed model valid?*

a. If there is an existing system, then compare model and system (from Step 2) performance measures for the existing system (see Sec. 5.4.5).

b. Regardless of whether there is an existing system, the simulation analysts and SMEs should review the model results for correctness.

c. Use sensitivity analyses (see Sec. 5.4.4) to determine what model factors have a significant impact on performance measures and, thus, have to be modeled carefully.

7. *Design experiments.*

a. Specify the following for each system configuration of interest:
- Length of each run
- Length of the warmup period, if one is appropriate
- Number of independent simulation runs using different random numbers (see Chap. 7)—facilitates construction of confidence intervals

8. *Make production runs.*

a. Production runs are made for use in Step 9.

9. *Analyze output data.*

a. Two major objectives in analyzing output data are:
- Determining the absolute performance of certain system configurations (see Chap. 9)
- Comparing alternative system configurations in a relative sense (see Chap. 10 and Sec. 11.2)

10. *Document, present, and use results.*

a. Document assumptions (see Step 2), computer program, and study's results for use in the current and future projects.

b. Present study's results.
- Use animation (see Chap. 3) to communicate model to managers and other people who are not familiar with all of the model details.
- Discuss model building and validation process to promote credibility.

c. Results are used in decision-making process if they are *both* valid and credible.

1.8 OTHER TYPES OF SIMULATION

Although the emphasis in this book is on discrete-event simulation, several other types of simulation are of considerable importance. Our goal here is to explain these other types of simulation briefly and to contrast them with discrete-event simulation. In particular, we shall discuss continuous, combined discrete-continuous, and Monte Carlo simulations.

1.8.1 Continuous Simulation

Continuous simulation concerns the modeling over time of a system by a representation in which the state variables change continuously with respect to time. Typically, continuous simulation models involve differential equations that give relationships for the rates of change of the state variables with time. If the differential equations are particularly simple, they can be solved analytically to give the values of the state variables for all values of time as a function of the values of the state variables at time 0. For most continuous models analytic solutions are not possible, however, and numerical-analysis techniques, e.g., Runge-Kutta integration, are used to integrate the differential equations numerically, given specific values for the state variables at time 0.

Several simulation products such as SIMULINK and Dymola, have been specifically designed for building continuous simulation models. In addition, the discrete-event simulation packages Arena [see Pegden, Shannon, and Sadowski (1995)], AweSim [Pritsker and O'Reilly (1999)], and Extend [Imagine (1997b)] have continuous modeling capabilities. These three simulation packages have the added advantage of allowing both discrete and continuous components simultaneously in one model (see Sec. 1.8.2). Readers interested in applications of continuous simulation may wish to consult the journal *Simulation*.

EXAMPLE 1.3. We now consider a continuous model of competition between two populations. Biological models of this type, which are called *predator-prey* (or *parasite-host*) models, have been considered by many authors, including Braun (1975, p. 583) and Gordon (1978, p. 103). An environment consists of two populations, predators and prey, which interact with each other. The prey are passive, but the predators depend on the prey as their source of food. [For example, the predators might be sharks and the prey might be food fish; see Braun (1975).] Let $x(t)$ and $y(t)$ denote, respectively, the numbers of individuals in the prey and predator populations at time t. Suppose that there is an ample supply of food for the prey and, in the absence of predators, that their rate of growth is $rx(t)$ for some positive r. (We can think of r as the natural birth rate minus the natural death rate.) Because of the interaction between predators and prey, it is reasonable to assume that the death rate of the prey due to interaction is proportional to the product of the two population sizes, $x(t)y(t)$. Therefore, the overall rate of change of the prey population, dx/dt, is given by

$$\frac{dx}{dt} = rx(t) - ax(t)y(t) \tag{1.8}$$

where a is a positive constant of proportionality. Since the predators depend on the prey for their very existence, the rate of change of the predators in the absence of prey is $-sy(t)$ for some positive s. Furthermore, the interaction between the two populations causes the predator population to increase at a rate that is also proportional to $x(t)y(t)$. Thus, the overall rate of change of the predator population, dy/dt, is

$$\frac{dy}{dt} = -sy(t) + bx(t)y(t) \tag{1.9}$$

where b is a positive constant. Given initial conditions $x(0) > 0$ and $y(0) > 0$, the solution of the model given by Eqs. (1.8) and (1.9) has the interesting property that $x(t) > 0$ and $y(t) > 0$ for all $t \geq 0$ [see Braun (1975)]. Thus, the prey population can never be completely extinguished by the predators. The solution $\{x(t), y(t)\}$ is also a periodic function of time. That is, there is a $T > 0$ such that $x(t + nT) = x(t)$ and $y(t + nT) = y(t)$ for all positive integers n. This result is not unexpected. As the predator population increases, the prey population decreases. This causes a decrease in the rate of increase of the predators, which eventually results in a decrease in the number of predators. This in turn causes the number of prey to increase, etc.

Consider the particular values $r = 0.001$, $a = 2 \times 10^{-6}$, $s = 0.01$, $b = 10^{-6}$ and the initial population sizes $x(0) = 12{,}000$ and $y(0) = 600$. Figure 1.69 is a numerical solution of Eqs. (1.8) and (1.9) resulting from using a computer package designed to solve systems of differential equations numerically (not explicitly a continuous simulation language).

Note that the above example was completely deterministic; i.e., it contained no random components. It is possible, however, for a continuous simulation model to

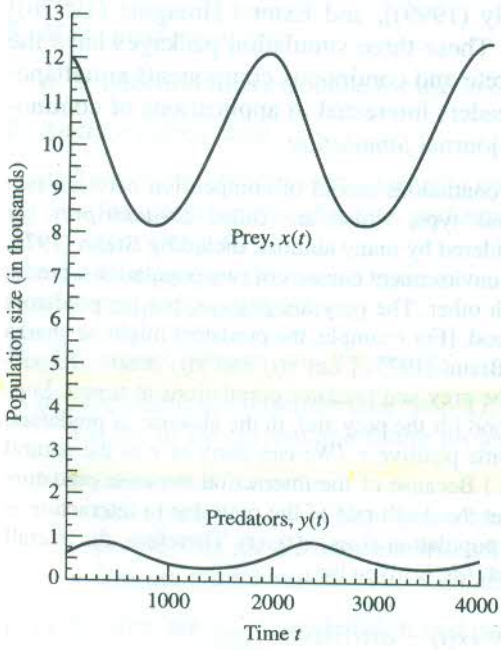

FIGURE 1.69
Numerical solution of a predator-prey model.

embody uncertainty; in Example 1.3 there could have been random terms added to Eqs. (1.8) and (1.9) that might depend on time in some way, or the constant factors could be modeled as quantities that change their value randomly at certain points in time.

1.8.2 Combined Discrete-Continuous Simulation

Since some systems are neither completely discrete nor completely continuous, the need may arise to construct a model with aspects of both discrete-event and continuous simulation, resulting in a *combined discrete-continuous* simulation. Pritsker (1995, pp. 61–62) describes the three fundamental types of interactions that can occur between discretely changing and continuously changing state variables:

- A discrete event may cause a discrete change in the value of a continuous state variable.
- A discrete event may cause the relationship governing a continuous state variable to change at a particular time.
- A continuous state variable achieving a threshold value may cause a discrete event to occur or to be scheduled.

Combined discrete-continuous simulation models can be built in Arena [Pegden, Shannon, and Sadowski (1995)], AweSim [Pritsker and O'Reilly (1999)], and Extend [Imagine (1997b)].

The following example of a combined discrete-continuous simulation is a brief description of a model described in detail by Pritsker (1995, pp. 354–364), who also provides other examples of this type of simulation.

EXAMPLE 1.4. Tankers carrying crude oil arrive at a single unloading dock, supplying a storage tank that in turn feeds a refinery through a pipeline. An unloading tanker delivers oil to the storage tank at a specified constant rate. (Tankers that arrive when the dock is busy form a queue.) The storage tank supplies oil to the refinery at a different specified rate. The dock is open from 6 A.M. to midnight and, because of safety considerations, unloading of tankers ceases when the dock is closed.

The discrete events for this (simplified) model are the arrival of a tanker for unloading, closing the dock at midnight, and opening the dock at 6 A.M. The levels of oil in the unloading tanker and in the storage tank are given by continuous state variables whose rates of change are described by differential equations [see Pritsker (1995, pp. 354–364) for details]. Unloading the tanker is considered complete when the level of oil in the tanker is less than 5 percent of its capacity, but unloading must be temporarily stopped if the level of oil in the storage tank reaches its capacity. Unloading can be resumed when the level of oil in the tank decreases to 80 percent of its capacity. If the level of oil in the tank ever falls below 5000 barrels, the refinery must be shut down temporarily. To avoid frequent startups and shutdowns of the refinery, the tank does not resume supplying oil to the refinery until the tank once again contains 50,000 barrels. Each of the five events concerning the levels of oil, e.g., the level of oil in the tanker falling below 5 percent of the tanker's capacity, is what Pritsker calls a *state event*. Unlike discrete events, state events are not scheduled but occur when a continuous state variable crosses a threshold.

1.8.3 Monte Carlo Simulation

We define *Monte Carlo* simulation to be a scheme employing random numbers, that is, U(0, 1) random variates, which is used for solving certain stochastic or deterministic problems where the passage of time plays no substantive role. Thus, Monte Carlo simulations are generally static rather than dynamic. The reader should note that although some authors define Monte Carlo simulation to be *any* simulation involving the use of random numbers, our definition is more restrictive. The name "Monte Carlo" simulation or method originated during World War II, when this approach was applied to problems related to the development of the atomic bomb. For a more detailed discussion of Monte Carlo simulation, see Hammersley and Handscomb (1964), Halton (1970), Rubinstein (1981, 1992), Morgan (1984), Fishman (1996), and Rubinstein, Melamed, and Shapiro (1998).

EXAMPLE 1.5. Suppose that we want to evaluate the integral

$$I = \int_a^b g(x)\,dx$$

where $g(x)$ is a real-valued function that is not analytically integrable. (In practice, Monte Carlo simulation would probably not be used to evaluate a single integral, since there are more efficient numerical-analysis techniques for this purpose. It is more likely to be used on a multiple-integral problem with an ill-behaved integrand.) To see how this *deterministic* problem can be approached by Monte Carlo simulation, let Y be the random variable $(b - a)g(X)$, where X is a continuous random variable distributed uniformly on $[a, b]$ [denoted by U(a, b)]. Then the expected value of Y is

$$\begin{aligned} E(Y) &= E[(b - a)g(X)] \\ &= (b - a)E[g(X)] \\ &= (b - a)\int_a^b g(x) f_X(x)\,dx \\ &= (b - a)\frac{\int_a^b g(x)\,dx}{b - a} \\ &= I \end{aligned}$$

where $f_X(x) = 1/(b - a)$ is the probability density function of a U(a, b) random variable (see Sec. 6.2.2). [For justification of the third equality, see, for example, Ross (1997, p. 42.] Thus, the problem of evaluating the integral has been reduced to one of estimating the expected value $E(Y)$. In particular, we shall estimate $E(Y) = I$ by the sample mean

$$\bar{Y}(n) = \frac{\sum_{i=1}^{n} Y_i}{n} = (b - a)\frac{\sum_{i=1}^{n} g(X_i)}{n}$$

where $X_1, X_2, \ldots, X_n$ are IID U(a, b) random variables. {It is instructive to think of $\bar{Y}(n)$ as an estimate of the area of the rectangle that has a base of length $(b - a)$ and a height $I/(b - a)$, which is the continuous average of $g(x)$ over $[a, b]$.} Furthermore, it can be shown that $E[\bar{Y}(n)] = I$, that is, $\bar{Y}(n)$ is an unbiased estimator of I, and

TABLE 1.3
$\bar{Y}(n)$ for various values of n resulting from applying Monte Carlo simulation to the estimation of the integral $\int_0^{\pi} \sin x\, dx = 2$

n	10	20	40	80	160
$\bar{Y}(n)$	2.213	1.951	1.948	1.989	1.993

$\text{Var}[\bar{Y}(n)] = \text{Var}(Y)/n$ (see Sec. 4.4). Assuming that $\text{Var}(Y)$ is finite, it follows that $\bar{Y}(n)$ will be arbitrarily close to I for sufficiently large n (with probability 1) (see Sec. 4.6).

To illustrate the above scheme numerically, suppose that we would like to evaluate the integral

$$I = \int_0^{\pi} \sin x\, dx$$

which can be shown by elementary calculus to have a value of 2. Table 1.3 shows the results of applying Monte Carlo simulation to the estimation of this integral for various values of n.

Monte Carlo simulation is now widely used to solve certain problems in statistics that are not analytically tractable. For example, it has been applied to estimate the critical values or the power of a new hypothesis test. Determining the critical values for the Kolmogorov-Smirnov test for normality, discussed in Sec. 6.6, is such an application. The advanced reader might also enjoy perusing the technical journals *Communications in Statistics* (Part B, Simulation and Computation), *Journal of Statistical Computation and Simulation*, and *Technometrics*, all of which contain many examples of this type of Monte Carlo simulation.

Finally, the procedures discussed in Sec. 9.4 can be used to determine the sample size n required to obtain a specified precision in a Monte Carlo simulation study.

1.9 ADVANTAGES, DISADVANTAGES, AND PITFALLS OF SIMULATION

We conclude this introductory chapter by listing some good and bad characteristics of simulation (as opposed to other methods of studying systems), and by noting some common mistakes made in simulation studies that can impair or even ruin a simulation project. This subject was also discussed to some extent in Sec. 1.2, but now that we have worked through some simulation examples, it is possible to be more specific.

As mentioned in Sec. 1.2, simulation is a widely used and increasingly popular method for studying complex systems. Some possible advantages of simulation that may account for its widespread appeal are the following.

- Most complex, real-world systems with stochastic elements cannot be accurately described by a mathematical model that can be evaluated *analytically*. Thus, a simulation is often the only type of investigation possible.

- Simulation allows one to estimate the performance of an existing system under some projected set of operating conditions.
- Alternative proposed system designs (or alternative operating policies for a single system) can be compared via simulation to see which best meets a specified requirement.
- In a simulation we can maintain much better control over experimental conditions than would generally be possible when experimenting with the system itself (see Chap. 11).
- Simulation allows us to study a system with a long time frame—e.g., an economic system—in compressed time, or alternatively to study the detailed workings of a system in expanded time.

Simulation is not without its drawbacks. Some disadvantages are as follows.

- Each run of a *stochastic* simulation model produces only *estimates* of a model's true characteristics for a particular set of input parameters. Thus, several independent runs of the model will probably be required for each set of input parameters to be studied (see Chap. 9). For this reason, simulation models are generally not as good at optimization as they are at comparing a fixed number of specified alternative system designs. On the other hand, an analytic model, *if appropriate,* can often easily produce the *exact* true characteristics of that model for a variety of sets of input parameters. Thus, if a "valid" analytic model is available or can easily be developed, it will generally be preferable to a simulation model.
- Simulation models are often expensive and time-consuming to develop.
- The large volume of numbers produced by a simulation study or the persuasive impact of a realistic animation (see Sec. 3.4.3) often creates a tendency to place greater confidence in a study's results than is justified. If a model is not a "valid" representation of a system under study, the simulation results, no matter how impressive they appear, will provide little useful information about the actual system.

When deciding whether or not a simulation study is appropriate in a given situation, we can only advise that these advantages and drawbacks be kept in mind and that all other relevant facets of one's particular situation be brought to bear as well. Finally, note that in some studies both simulation and analytic models might be useful. In particular, simulation can be used to check the validity of assumptions needed in an analytic model. On the other hand, an analytic model can suggest reasonable alternatives to investigate in a simulation study.

Assuming that a decision has been made to use simulation, we have found the following pitfalls to the successful completion of a simulation study [see also Law and McComas (1989)]:

- Failure to have a well-defined set of objectives at the beginning of the simulation study
- Inappropriate level of model detail
- Failure to communicate with management throughout the course of the simulation study
- Misunderstanding of simulation by management
- Treating a simulation study as if it were primarily an exercise in computer programming

- Failure to have people with a knowledge of simulation methodology (Chaps. 5, 6, 9, etc.) and statistics on the modeling team
- Failure to collect good system data
- Inappropriate simulation software
- Obliviously using simulation software products whose complex macro statements may not be well documented and may not implement the desired modeling logic
- Belief that easy-to-use simulation packages, which require little or no programming, require a significantly lower level of technical competence
- Misuse of animation
- Failure to account correctly for sources of randomness in the actual system
- Using arbitrary distributions (e.g., normal, uniform, or triangular) as input to the simulation
- Analyzing the output data from one simulation run (replication) using formulas that assume independence
- Making a single replication of a particular system design and treating the output statistics as the "true answers"
- Comparing alternative system designs on the basis of one replication for each design
- Using the wrong performance measures

We will have more to say about what *to* do (rather than what *not* to do) in the remaining chapters of this book.

APPENDIX 1A
FIXED-INCREMENT TIME ADVANCE

As mentioned in Sec. 1.3.1, the second principal approach for advancing the simulation clock in a discrete-event simulation model is called *fixed-increment time advance*. With this approach, the simulation clock is advanced in increments of exactly Δt time units for some appropriate choice of Δt. After each update of the clock, a check is made to determine if any events should have occurred during the previous interval of length Δt. If one or more events were scheduled to have occurred during this interval, these events are considered to occur at the *end* of the interval and the system state (and statistical counters) are updated accordingly. The fixed-increment time-advance approach is illustrated in Fig. 1.70, where the curved arrows represent the advancing of the simulation clock and e_i ($i = 1, 2, \ldots$) is the *actual* time of occurrence of the ith event of any type (*not* the ith value of the simulation clock). In the time interval $[0, \Delta t)$, an event occurs at time e_1 but is considered to occur at time

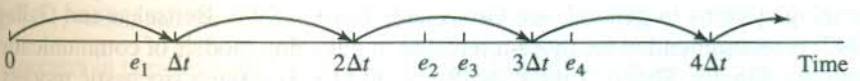

FIGURE 1.70
Illustration of fixed-increment time advance.