5 Simulation Modeling

Introduction

In many situations a modeler is unable to construct an analytic (symbolic) model adequately explaining the behavior being observed because of its complexity or the intractability of the proposed explicative model. Yet if it is necessary to make predictions about the behavior, the modeler may conduct experiments (or gather data) to investigate the relationship between the dependent variable(s) and selected values of the independent variable(s) within some range. We constructed empirical models based on collected data in Chapter 4. To collect the data, the modeler may observe the behavior directly. In other instances, the behavior might be duplicated (possibly in a scaled-down version) under controlled conditions, as will be done when predicting the size of craters in Section 8.4.

In some circumstances, it may not be feasible either to observe the behavior directly or to conduct experiments. For instance, consider the service provided by a system of elevators during morning rush hour. After identifying an appropriate problem and defining what is meant by good service, we might suggest some alternative delivery schemes, such as assigning elevators to even and odd floors or using express elevators. Theoretically, each alternative could be tested for some period of time to determine which one provided the best service for particular arrival and destination patterns of the customers. However, such a procedure would probably be very disruptive because it would be necessary to harass the customers constantly as the required statistics are collected. Moreover, the customers would become very confused because the elevator delivery system would keep changing. Another problem concerns testing alternative schemes for controlling automobile traffic in a large city. It would be impractical to constantly change directions of the one-way streets and the distribution of traffic signals to conduct tests.

In still other situations, the system for which alternative procedures need to be tested *may not even exist yet.* An example is the situation of several proposed communications networks with the problem of determining which is best for a given office building. Still another example is the problem of determining locations of machines in a new industrial plant. The *cost* of conducting experiments may be prohibitive. This is the case when trying to predict the effects of various alternatives for protecting and evacuating the population in case of failure of a nuclear power plant.

In instances in which the behavior cannot be explained analytically or data collected directly the modeler might *simulate* the behavior indirectly in some man-

ner and then test the various alternatives under consideration to estimate how each affects the behavior. Data can then be collected to determine which alternative is best. An example is to determine the drag force on a proposed submarine. Because it is infeasible to build a prototype, we can build a scaled model to simulate the behavior of the actual submarine. Another example of this type of simulation is using a scaled model of a jet airplane in a wind tunnel to estimate the effects of very high speeds for various designs of the aircraft. There is yet another type of simulation, which we will study in this chapter. The method for simulating behavior is called **Monte Carlo simulation** and is typically accomplished with the aid of a computer.

Suppose we are investigating the service provided by a system of elevators at morning rush hour. In Monte Carlo simulation the arrival of customers at the elevators during the hour and the destination floors they select need to be replicated. That is, the distribution of arrival times and the distribution of floors desired on the simulated trial must portray a possible rush hour. Moreover, after we simulated many trials, the daily distribution of arrivals and destinations that occur must mimic the real-world distributions in proper proportions. When we are satisfied that the behavior is adequately duplicated, we can investigate various alternative strategies for operating the elevators. Using a large number of trials, we can gather appropriate statistics, such as the average total delivery time of a customer or the length of the longest queue. These statistics can help determine the best strategy for operating the elevator system.

This chapter provides a brief introduction to Monte Carlo simulation. Additional studies in probability and statistics are required to delve into the intricacies of computer simulation and understand its appropriate uses. Nevertheless, you will gain some appreciation of this powerful component of mathematical modeling. Keep in mind that there is a danger in placing too much confidence in the predictions resulting from a simulation, especially if the assumptions inherent in the simulation are not clearly stated. Moreover, the appearance of using large amounts of data and huge amounts of computer time, coupled with the fact the layman can understand a simulation model and computer output with relative ease, often leads to overconfidence in the results.

When any Monte Carlo simulation is performed, random numbers are used. We discuss how to generate random numbers in Section 5.2. Loosely speaking, a "sequence of random numbers uniformly distributed in an interval m to n" is a set of numbers with no apparent pattern, where each number between m and n can appear with equal likelihood. For example, if you toss a six-sided die 100 times and write down the number showing on the die each time, you will have written down a sequence of 100 random integers approximately uniformly distributed over the interval 1 to 6. Now, suppose that random numbers consisting of six digits can be generated. The tossing of a coin can be duplicated by generating a random number and assigning it a head if the random number is even and a tail if the random number is odd. If this trial is replicated a large number of times, you would expect heads to occur about 50% of the time. However, there is an element of chance involved. It is possible that a run of 100 trials could produce 51 heads and that the next 10 trials (although not very likely) would produce all heads. Thus, the estimate with 110 trials is actually worse than the estimate with 100 trials. Processes with an

element of chance involved are called **probabilistic**, as opposed to **deterministic**, processes. Monte Carlo simulation is therefore a probabilistic model.

The modeled behavior may be either deterministic or probabilistic. For instance, the area under a curve is deterministic (even though it may be impossible to find it precisely). On the other hand, the time between arrivals of customers at the elevator on a particular day is probabilistic behavior. Referring to Figure 5.1 we can see that a deterministic model can be used to approximate either a deterministic or probabilistic behavior and, likewise, a Monte Carlo simulation can be used to approximate a deterministic behavior (as you will see with a Monte Carlo approximation to an area under a curve) or a probabilistic one. However, as we would expect, the real power of Monte Carlo simulation lies in modeling a probabilistic behavior.

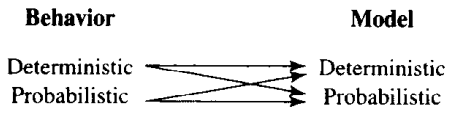


Figure 5.1
The behavior and the model can be either deterministic or probabilistic

A principal advantage of Monte Carlo simulation is the relative ease with which it can sometimes be used to approximate very complex probabilistic systems. Additionally, Monte Carlo simulation provides performance estimation over a wide range of conditions rather than a very restricted range as often required by an analytic model. Furthermore, because a particular submodel can be changed rather easily in a Monte Carlo simulation (such as the arrival and destination patterns of customers at the elevators), there is the potential of conducting a sensitivity analysis. Still another advantage is that the modeler has control over the level of detail in a simulation. For example, a very long time frame can be compressed or a small time frame expanded, giving a great advantage over experimental models. Finally, there are very powerful, high-level simulation languages (such as GPSS, GASP, PROLOG, SIMAN, SLAM, and DYNAMO) that eliminate much of the tedious labor in constructing a simulation model.

On the negative side, simulation models are typically expensive to develop and operate. They may require many hours to construct and large amounts of computer time and memory to run. Another disadvantage is that the probabilistic nature of the simulation model limits the conclusions that can be drawn from a particular run unless a sensitivity analysis is conducted. Such an analysis often requires many more runs just to consider a small number of combinations of conditions that can occur in the various submodels. This limitation then forces the modeler to estimate which combination might occur for a particular set of conditions.

5.1 Simulating Deterministic Behavior: Area Under a Curve

In this section we illustrate the use of Monte Carlo simulation to model a deterministic behavior, the area under a curve. We begin by finding an approximate value to the area under a nonnegative curve. Specifically, suppose $y = f(x)$ is some given

continuous function satisfying $0 \leq f(x) \leq M$ over the closed interval $a \leq x \leq b$. Here, the number M is simply some constant that *bounds* the function. This situation is depicted in Figure 5.2. Notice that the area we seek is wholly contained within the rectangular region of height M and length $b-a$ (the length of the interval over which f is defined).

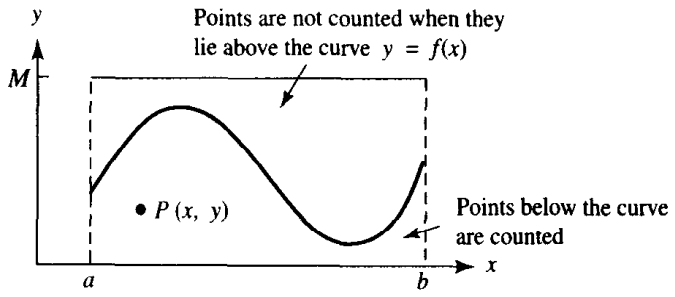


Figure 5.2
The area under the nonnegative curve $y = f(x)$ over $a \leq x \leq b$ is contained within the rectangle of height M and base length $b - a$

Now we select a point $P(x, y)$ at random from within the rectangular region. We will do so by generating two random numbers, x and y, satisfying $a \leq x \leq b$ and $0 \leq y \leq M$, and interpreting them as a point P with coordinates x and y. Once $P(x, y)$ is selected, we ask if it lies within the region below the curve; that is, does the y-coordinate satisfy $0 \leq y \leq f(x)$? If the answer is yes, then count the point P by adding one to some counter. Two counters will be necessary: one to count the total points generated and a second to count those points that lie below the curve (Figure 5.2). You can then calculate an approximate value for the area under the curve by the following formula:

$$\frac{\text{area under curve}}{\text{area of rectangle}} \approx \frac{\text{number of points counted below curve}}{\text{total number of random points}}$$

As discussed in the Introduction, the Monte Carlo technique is probabilistic and typically requires a large number of trials before the deviation between the predicted and true values becomes small. A discussion of the number of trials needed to ensure a predetermined level of confidence in the final estimate requires a background in statistics. However, as a general rule, to double the accuracy of the result (i.e., cut the expected error in half), about four times as many experiments are necessary.

The following algorithm gives the sequence of calculations needed for a general computer simulation of this Monte Carlo technique for finding the area under a curve:

Monte Carlo Area Algorithm

Input Total number n of random points to be generated in the simulation.

Output AREA = approximate area under the specified curve $y = f(x)$ over the given interval $a \leq x \leq b$, where $0 \leq f(x) < M$.

Step 1 Initialize: COUNTER = 0.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3–5.

Step 3 Calculate random coordinates x_i and y_i, satisfying $a \leq x_i \leq b$ and $0 \leq y_i < M$.

Step 4 Calculate $f(x_i)$ for the random x_i coordinate.

Step 5 If $y_i \leq f(x_i)$, then increment the COUNTER by 1. Otherwise, leave COUNTER as is.

Step 6 Calculate AREA $= M(b - a)$ COUNTER$/n$.

Step 7 OUTPUT (AREA)
STOP

Table 5.1 gives the results of several different simulations to obtain the area beneath the curve $y = \cos x$ over the interval $-\pi/2 \leq x \leq \pi/2$, where $0 \leq \cos x < 2$.

Table 5.1 Monte Carlo approximation to the area under the curve $y = \cos x$ over the interval $-\pi/2 \leq x \leq \pi/2$

Number of points	Approximation to area	Number of points	Approximation to area
100	2.07345	2000	1.94465
200	2.13628	3000	1.97711
300	2.01064	4000	1.99962
400	2.12058	5000	2.01429
500	2.04832	6000	2.02319
600	2.09440	8000	2.00669
700	2.02857	10000	2.00873
800	1.99491	15000	2.00978
900	1.99666	20000	2.01093
1000	1.96664	30000	2.01186

The actual area under the curve $y = \cos x$ over the given interval is 2 square units. Note that even with the relatively large number of points generated, the error is significant. For functions of one variable, the Monte Carlo technique is generally not competitive with quadrature techniques that you will learn in numerical analysis. The lack of an error bound and the difficulty in finding an upper bound M are disadvantages as well. Nevertheless, the Monte Carlo technique can be extended to functions of several variables and becomes more practical in that situation.

Volume Under a Surface

Let's consider finding part of the volume of the sphere

$$x^2 + y^2 + z^2 \leq 1$$

that lies in the first octant, $x > 0$, $y > 0$, $z > 0$ (Figure 5.3).

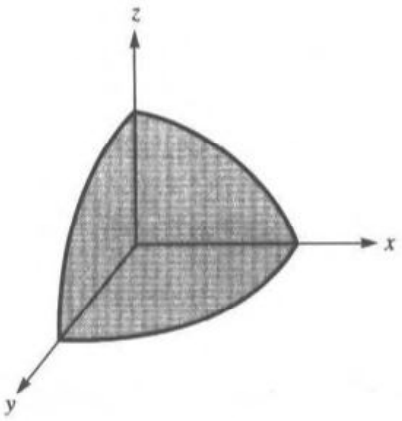


Figure 5.3
Volume of a sphere $x^2 + y^2 + z^2 \leq 1$ that lies in the first octant, $x > 0$, $y > 0, z > 0$

The methodology to approximate the volume is very similar to that of finding the area under a curve. However, now we will use an approximation for the volume under the surface by the following rule:

$$\frac{\text{volume under surface}}{\text{volume of box}} \approx \frac{\text{number of points counted below surface in 1st octant}}{\text{total number of points}}$$

The following algorithm gives the sequence of calculations required to employ Monte Carlo techniques to find the approximate volume of the region:

Monte Carlo Volume Algorithm

Input Total number n of random points to be generated in the simulation.

Output VOLUME = approximate volume enclosed by the specified function, $z = f(x, y)$ in the first octant, $x > 0$, $y > 0$, $z > 0$.

Step 1 Initialize: COUNTER = 0.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3–5.

Step 3 Calculate random coordinates x_i, y_i, z_i that satisfy: $0 \leq x_i \leq 1$, $0 \leq y_i \leq 1$, $0 \leq z_i \leq 1$.
(In general, $a \leq x_i \leq b, c \leq y_i \leq d, 0 \leq z_i \leq M$.)

Step 4 Calculate $f(x_i, y_i)$ for the random coordinate (x_i, y_i).

Step 5 If random $z_i \leq f(x_i, y_i)$, then increment the COUNTER by 1. Otherwise, leave COUNTER as is.

Step 6 Calculate VOLUME $= M(d - c)(b - a)$COUNTER$/n$.

Step 7 OUTPUT (VOLUME)
STOP

Table 5.2 gives the results of several Monte Carlo runs to obtain the approximate volume of

$$x^2 + y^2 + z^2 \leq 1$$

that lies in the first octant, $x > 0$, $y > 0$, $z > 0$.

Table 5.2 Monte Carlo approximation to the volume in the first octant under the surface $x^2 + y^2 + z^2 \leq 1$

Number of points	Approximate volume
100	0.4700
200	0.5950
300	0.5030
500	0.5140
1,000	0.5180
2,000	0.5120
5,000	0.5180
10,000	0.5234
20,000	0.5242

The actual volume in the first octant is found to be approximately 0.5236 cubic units ($\pi/6$ with $R = 1$). Generally, though not uniformly, the error becomes smaller as the number of points generated increases.

5.1 Problems

1. Each ticket in a lottery contains a single "hidden" number according to the following scheme: 55% of the tickets contain a 1, 35% contain a 2, and 10% contain a 3. A participant in the lottery wins a prize by obtaining all three numbers 1, 2, and 3. Describe an experiment that could be used to determine how many tickets you would expect to buy to win a prize.

2. Two record companies, A and B, produce classical music recordings. Label A is a budget label and 5% of A's new compact discs exhibit significant degrees of warpage. Label B is manufactured under tighter quality control (and consequently more expensive) than A, so only 2% of its compact discs are warped. You purchase one label A and one label B recording at your local store on a regular basis. Describe an experiment that could be used to determine how many times you would expect to make such a purchase before buying two warped compact discs for a given sale.

3. Using Monte Carlo simulation, write an algorithm to calculate an approximation to π by considering the number of random points selected inside the quarter circle

$$Q : x^2 + y^2 = 1, x \geq 0, y \geq 0$$

where the quarter circle is taken to be inside the square

$$S: 0 \le x \le 1 \text{ and } 0 \le y \le 1$$

Use the equation that $\pi/4 =$ area Q/area S.

4. Using Monte Carlo simulation, write an algorithm to calculate that part of the volume of an ellipsoid

$$\frac{x^2}{2} + \frac{y^2}{4} + \frac{z^2}{8} \le 16$$

that lies in the first octant, $x > 0, y > 0, z > 0$.

5. Using Monte Carlo simulation, write an algorithm to calculate the volume trapped between the two paraboloids

$$z = 8 - x^2 - y^2 \quad \text{and} \quad z = x^2 + 3y^2$$

Note that the two paraboloids intersect on the elliptic cylinder

$$x^2 + 2y^2 = 4$$

5.2 Generating Random Numbers

In the previous section, we developed algorithms for Monte Carlo simulations to find areas and volumes. A key ingredient common to each algorithm is the need for random numbers. Random numbers have a variety of applications, including gambling problems, finding an area or volume, and modeling larger complex systems such as large-scale combat operations or air traffic control situations.

In some sense a computer really does not generate random numbers because computers employ deterministic algorithms. However, we can generate sequences of pseudorandom numbers that, for all practical purposes, may be considered random. There is no single best random number generator or best test to ensure randomness.

There are complete courses of study for random numbers and simulations that cover in-depth the methods and tests for pseudorandom number generators. Our purpose here is to introduce a few random number methods that can be utilized to generate sequences of numbers that are nearly random.

Many programming languages, such as Pascal and Basic, and other software (e.g., Minitab, MATLAB, and EXCEL) have built-in random number generators for user convenience.

Middle-Square Method

The middle-square method was developed in 1946 by John Von Neuman, S. Ulm, and N. Metropolis at Los Alamos Laboratories to simulate neutron collisions as part of the Manhattan Project. Their middle-square method works as follows:

1. Start with a four-digit number x_0, called the *seed.*
2. Square it to obtain an eight-digit number (add a leading zero if necessary).
3. Take the middle four digits as the next random number.

Continuing in this manner, we obtain a sequence that appears to be random over the integers from 0 to 9999. These integers can then be scaled to any interval a to b. For example, if we wanted numbers from 0 to 1 we would divide the four-digit numbers by 10,000. Let's illustrate the middle-square method.

Pick a seed, say $x_0 = 2041$ and square it (adding a leading zero) to get 04165681. The middle four digits give the next random number, 1656. Generating 9 random numbers in this way yields

n	0	1	2	3	4	5	6	7	8	9	10	11	12
x_n	2041	1656	7423	1009	0180	0324	1049	1004	80	64	40	16	2

We can use more than 4 digits if we wish, but we always take the middle number of digits equal to the number of digits in the seed. For example, if $x_0 = 653217$ (6 digits), its square 426,692,449,089 has 12 digits. Thus, take the middle 6 digits as the random number, namely, 692449.

The middle-square method is reasonable, but it has a major drawback in its tendency to degenerate to zero (where it will stay forever). With the seed 2041, the random sequence does seem to be approaching zero. How many numbers can be generated until we are almost at zero?

Linear Congruence

The linear congruence method was introduced by D. H. Lehmer in 1951 and a majority of pseudorandom numbers used today are based on this method. One advantage it has over other methods is that seeds can be selected that generate patterns that eventually cycle (we illustrate this concept with an example). However, the length of the cycle is so large that the pattern does not repeat itself on large computers for most applications. The method requires the choice of three integers: a, b, and c. Given some initial seed, say x_0, we generate a sequence by the rule

$$x_{n+1} = (a \times x_n + b)\text{mod}(c)$$

where c is the modulus, a is the multiplier, and b is the increment. The qualifier mod(c) in the equation means to obtain the remainder after dividing the quantity $(a \times x_n + b)$ by c. For example, with $a = 1$, $b = 7$, and $c = 10$,

$$x_{n+1} = (1 \times x_n + 7)\text{mod}(10)$$

means x_{n+1} is the integer remainder upon dividing $x_n + 7$ by 10. Thus, if $x_n = 115$, then $x_{n+1} = \textit{remainder}\ (\frac{122}{10}) = 2$.

Before investigating the linear congruence methodology, we need to discuss **cycling**, which is a major problem that occurs with random numbers. Cycling

means the sequence repeats itself, and although undesirable, it is unavoidable. At some point, all pseudorandom number generators begin to cycle. Let's illustrate cycling with an example.

If we set our seed at $x_0 = 7$, we find $x_1 = (1 \times 7 + 7) \bmod(10)$ or $14 \bmod(10)$, which is 4. Repeating this same procedure, we obtain the following sequence:

$$7, 4, 1, 8, 5, 2, 9, 6, 3, 0, 7, 4, \ldots$$

and the original sequence repeats again and again. Note that there is cycling after 10 numbers. The methodology produces a sequence of integers between 0 and $c - 1$ inclusively before cycling (which includes the possible remainders after dividing the integers by c). Cycling is guaranteed with at most c numbers in the random number sequence. Nevertheless, c can be chosen to be very large and a and b chosen in such a way to obtain a full set of c numbers before cycling begins to occur. Many computers use $c = 2^{31}$ for the large value of c. Again, we can scale the random numbers to obtain a sequence between any limits a and b, as required.

A second problem that can occur with the linear congruence method is lack of statistical independence among the members in the list of random numbers. Any correlations between the nearest neighbors, the next-nearest neighbors, the third-nearest neighbors, and so forth are generally unacceptable. (Because we live in a three-dimensional world, third-nearest neighbor correlations can be particularly damaging in physical applications.) Pseudorandom number sequences can never be completely statistically independent because they are generated by a mathematical formula or algorithm. Nevertheless, the sequence will appear (for practical purposes) independent when it is subjected to certain statistical tests. These concerns are best addressed in a course in statistics.

5.2 Problems

1. Use the middle-square method to generate

(a) 10 random numbers using $x_0 = 1009$.

(b) 20 random numbers using $x_0 = 653217$.

(c) 15 random numbers using $x_0 = 3043$.

(d) Comment about the results of each sequence. Was there cycling? Did each sequence degenerate rapidly?

2. Use the linear congruence method to generate

(a) 10 random numbers using $a = 5$, $b = 1$, and $c = 8$.

(b) 15 random numbers using $a = 1$, $b = 7$, and $c = 10$.

(c) 20 random numbers using $a = 5$, $b = 3$, and $c = 16$.

(d) Comment about the results of each sequence. Was there cycling? If so, when did it occur?

5.2 Projects

1. Complete the requirement for UMAP module 269, "Monte Carlo: The Use of Random Digits to Simulate Experiments," by Dale T. Hoffman. The Monte Carlo technique is presented, explained, and used to find approximate solutions to several realistic problems. Simple experiments are included for student practice.

2. "Random Numbers" by Mark D. Myerson, UMAP 590. This module discusses methods for generating random numbers and presents tests for determining the randomness of a string of numbers. Complete this module and prepare a short report on testing for randomness.

3. Write a computer program to generate uniformly distributed random integers in the interval $m < x < n$, where m and n are integers, according to the following algorithm:

Step 1 Let $d = 2^{31}$ and choose N (the number of random numbers to generate).

Step 2 Choose any seed integer Y such that

$$999999 > Y > 100000$$

Step 3 Let $i = 1$.

Step 4 Let $Y = (15625\,Y + 22221) \bmod(d)$.

Step 5 Let $X_i = m + \text{floor}[(n - m + 1)Y/d]$.

Step 6 Increment i by 1: $i = i + 1$.

Step 7 Go to Step 4 unless $i = N + 1$.

Here, floor (p) means the largest integer not exceeding p.

For most choices of Y, the numbers $X_1, X_2, \ldots$ form a sequence of (pseudo)random integers as desired. One possible recommended choice is $Y = 568731$. To generate random numbers (not just integers) in an interval a to b with $a < b$, use the preceding algorithm, replacing the formula in Step 5 by

$$\text{Let } X_i = a + \frac{Y(b-a)}{d-1}$$

4. Write a program to generate 1000 integers between 1 and 5 in a random fashion so that 1 occurs 22% of the time, 2 occurs 15% of the time, 3 occurs 31% of the time, 4 occurs 26% of the time, and 5 occurs 6% of the time. Over what interval would you generate the random numbers? How do you decide which of the integers from 1 to 5 has been generated according to its specified chance of selection?

5. Write a program or use a spreadsheet to find the approximate area or volumes in Problems 3–5 in Section 5.1.

5.3 Simulating Probabilistic Behavior

One of the keys to good Monte Carlo simulation practices is an understanding of the axioms of probability. The term probability refers to the study of both randomness and uncertainty as well as the quantifying of the likelihoods associated with various outcomes. Probability can be seen as a long-term average. For example, if the probability of an event occurring is 1 out of 5, then in the long run, the chance of the event happening is 1/5. Over the long haul, the probability of an event can be thought of as the ratio of

$$\frac{\text{number of favorable events}}{\text{total number of events}}$$

Our goal in this section is to show how to model simple probabilistic behavior to build intuition and understanding before developing submodels of probabilistic processes to incorporate in simulations (Sections 5.4 and 5.5)

We examine three simple probabilistic models:

1. Flip of a fair coin
2. Roll of a fair die or pair of dice
3. Roll of an unfair die or pair of unfair dice

A Fair Coin

Most people realize that the chance of obtaining a head or a tail on a coin is 1/2. What happens if we actually start flipping a coin? Will one out of every two flips be a head? Probably not. Again, probability is a long-term average. Thus, in the long run, the ratio of heads to the number of flips approaches 0.5. Let's define $f(x)$ as follows, where x is a random number between [0, 1]:

$$f(x) = \begin{cases} \text{Head}, & 0 \leq x \leq 0.5 \\ \text{Tail}, & 0.5 < x \leq 1 \end{cases}$$

Note that $f(x)$ assigns the outcome head or tail to a number between [0, 1]. We want to take advantage of the cumulative nature of this function as we make random assignments to numbers between [0, 1]. In the long run we expect to find the following percent occurrences:

Random number interval	Cumulative occurrences	Percent occurrence
$x < 0$	0	0.00
$0 < x < 0.5$	0.5	0.50
$0.5 < x < 1.0$	1	0.50

Let's illustrate using the following algorithm:

Monte Carlo Fair Coin Algorithm

Input Total number n of random flips of a fair coin to be generated in the simulation.

Output Probability of getting a head when we flip a fair coin.

Step 1 Initialize: COUNTER = 0.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3 and 4.

Step 3 Obtain a random number x_i between 0 and 1.

Step 4 If $0 \leq x_i \leq 0.5$, then COUNTER = COUNTER + 1. Otherwise, leave COUNTER as is.

Step 5 Calculate $P(\text{head}) = \text{COUNTER}/n$.

Step 6 OUTPUT Probability of heads, $P(\text{head})$.
STOP

Table 5.3 illustrates our results for various choices n of the number of random x_i generated. Note that as n gets large, the probability of heads occurring is 0.5, or half the time.

Table 5.3 Results from flipping a fair coin

Number of flips	Number of heads	Percent heads
100	49	0.49
200	102	0.51
500	252	0.504
1,000	492	0.492
5,000	2469	0.4930
10,000	4993	0.4993

Roll of a Fair Die

Rolling a fair die adds a new twist to the process. In the flip of a coin, only one event is assigned (with two possible answers, yes or no). Now we must devise a method to assign six events because a die consists of the numbers {1, 2, 3, 4, 5, 6}. The probability of each event occurring is 1/6 because each number is equally likely to occur. As before, this probability of a particular number occurring is defined to be

$$\frac{\text{number of occurences of the particular number } \{1, 2, 3, 4, 5, 6\}}{\text{total number of trials}}$$

We can use the following algorithm to generate our experiment for a roll of a die:

Monte Carlo Roll of a Fair Die Algorithm

Input Total number n of random rolls of a die in the simulation.

Output The percentage or probability for rolls {1, 2, 3, 4, 5, 6}.

Step 1 Initialize COUNTER 1 through COUNTER 6 to zero.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3 and 4.

Step 3 Obtain a random number satisfying $0 \leq x_i \leq 1$.

Step 4 If x_i belongs to these intervals, then increment the appropriate COUNTER.

$0 \leq x_i \leq \frac{1}{6}$ COUNTER 1 = COUNTER 1 + 1

$\frac{1}{6} < x_i \leq \frac{2}{6}$ COUNTER 2 = COUNTER 2 + 1

$\frac{2}{6} < x_i \leq \frac{3}{6}$ COUNTER 3 = COUNTER 3 + 1

$\frac{3}{6} < x_i \leq \frac{4}{6}$ COUNTER 4 = COUNTER 4 + 1

$\frac{4}{6} < x_i \leq \frac{5}{6}$ COUNTER 5 = COUNTER 5 + 1

$\frac{5}{6} < x_i \leq 1$ COUNTER 6 = COUNTER 6 + 1

Step 5 Calculate probability of each roll $j = \{1, 2, 3, 4, 5, 6\}$ by COUNTER$(j)/n$.

Step 6 OUTPUT probabilities.
STOP

Table 5.4 illustrates the results for 10, 100, 1000, 10,000, and 100,000 runs. We see that with 100,000 runs we are close (for these trials) to the expected results.

Table 5.4 Results from a roll of a fair die (n = number of trials)

Die value	10	100	1000	10,000	100,000	Expected results
1	0.300	0.190	0.152	0.1703	0.1652	0.1667
2	0.00	0.150	0.152	0.1652	0.1657	0.1667
3	0.100	0.090	0.157	0.1639	0.1685	0.1667
4	0.00	0.160	0.180	0.1653	0.1685	0.1667
5	0.400	0.150	0.174	0.1738	0.1676	0.1667
6	0.200	0.160	0.185	0.1615	0.1652	0.1667

Roll of an Unfair Die

Let's consider a probability model in which each event is not equally likely. Assume the die is loaded or biased according to the following empirical distribution:

Roll value	*P(roll)*
1	0.1
2	0.1
3	0.2
4	0.3
5	0.2
6	0.1

The cumulative occurrences for the function to be used in our algorithm would be

Value of x_i	Assignment
[0, 0.1]	ONE
(0.1, 0.2]	TWO
(0.2, 0.4]	THREE
(0.4, 0.7]	FOUR
(0.7, 0.9]	FIVE
(0.9, 1.0]	SIX

We model the roll of an unfair die using the following algorithm:

Monte Carlo Roll of an Unfair Die Algorithm

Input Total number n of random rolls of a die in the simulation.

Output The percentage or probability for rolls $\{1, 2, 3, 4, 5, 6\}$.

Step 1 Initialize COUNTER 1 through COUNTER 6 to zero.

Step 2 For $i = 1, 2, \ldots, n$, do Steps 3 and 4.

Step 3 Obtain a random number satisfying $0 \leq x_i \leq 1$.

Step 4 If x_i belongs to these intervals, then increment the appropriate COUNTER.

$0 \leq x_i \leq 0.1$ COUNTER 1 = COUNTER 1 + 1

$0.1 < x_i \leq 0.2$ COUNTER 2 = COUNTER 2 + 1

$0.2 < x_i \leq 0.4$ COUNTER 3 = COUNTER 3 + 1

$0.4 < x_i \leq 0.7$ COUNTER 4 = COUNTER 4 + 1

$0.7 < x_i \leq 0.9$ COUNTER 5 = COUNTER 5 + 1

$0.9 < x_i \leq 1.0$ COUNTER 6 = COUNTER 6 + 1

Step 5 Calculate probability of each roll $j = \{1, 2, 3, 4, 5, 6\}$ by COUNTER$(j)/n$.

Step 6 OUTPUT probabilities.
STOP

The results are shown in Table 5.5. Note that a large number of trials are required for the model to approach the long-term probabilities.

In the next section, we will see how to use these ideas to simulate a real-world probabilistic situation.

Table 5.5 Results from a roll of an unfair die

Die value	100	1000	5000	10,000	40,000	Expected results
1	0.080	0.078	0.094	0.0948	0.0948	0.1
2	0.110	0.099	0.099	0.0992	0.0992	0.1
3	0.230	0.199	0.192	0.1962	0.1962	0.2
4	0.360	0.320	0.308	0.3082	0.3081	0.3
5	0.110	0.184	0.201	0.2012	0.2011	0.2
6	0.110	0.120	0.104	0.1044	0.1045	0.1

5.3 Problems

1. You arrive at the beach for a vacation and are dismayed to learn that the local weather station is predicting a 50% chance of rain every day. Using Monte Carlo simulation, predict the chance that it rains three consecutive days during your vacation.
2. Use Monte Carlo simulation to approximate the probability of three heads occurring when five fair coins are flipped.
3. Use Monte Carlo simulation to simulate the sum of 100 consecutive rolls of a fair die.
4. Given loaded dice according to the following distribution, use Monte Carlo simulation to simulate the sum of 300 rolls of two unfair dice.

Roll	Die 1	Die 2
1	0.1	0.3
2	0.1	0.1
3	0.2	0.2
4	0.3	0.1
5	0.2	0.05
6	0.1	0.25

5. Make up a game that uses a flip of a fair coin, and then use Monte Carlo simulation to predict the results of the game.

5.3 Projects

1. *Blackjack*—Construct and perform a Monte Carlo simulation of blackjack (also called twenty-one). The rules of blackjack are as follows:

   Most casinos use six or eight decks of cards when playing this game to inhibit "card counters." You will use two decks of cards in your simulation (104 cards total). There are only two players, you and the dealer. Each player receives two cards to begin play. The cards are worth their face value for 2–10, 10 for face cards (jack, queen, and king), and either 1 or 11 points for Aces. The object of the game is to obtain a total as close to 21 as possible without going over (called "busted") so that your total is more than the dealer's.

   If the first two cards total 21 (ace–10 or ace–face card), this is called blackjack and is an automatic winner (unless both you and the dealer have blackjack, in which case it is a tie, or "push," and your bet remains on the table). Winning via blackjack pays you 3 to 2, or 1.5 to 1 (a $1 bet reaps $1.50 and you do not lose the $1 you bet).

   If neither you nor the dealer has blackjack, you can take as many cards as you want, one at a time, to try to get as close to 21 as possible. If you go over 21, you

lose and the game ends. Once you are satisfied with your score, you "stand." The dealer then draws cards according to the following rules:

The dealer stands on 17, 18, 19, 20, or 21. The dealer must draw a card if the total is 16 or less. The dealer always counts aces as 11 unless it causes him or her to bust, in which case it is counted as a 1. For example, an ace–6 combo for the dealer is 17, not 7 (the dealer has no option), and the dealer must stand on 17. However, if the dealer has an ace–4 (for 15) and draws a king, then the new total is 15 because the ace reverts to its value of 1 (so as not to go over 21). The dealer would then draw another card.

If the dealer goes over 21, you win (even your bet money; you gain $1 for every $1 you bet). If the dealer's total exceeds your total, you lose all the money you bet. If the dealer's total equals your total, it is a push (no money exchanges hands; you do not lose your bet, but neither do you gain any money).

What makes the game exciting in a casino is that the dealer's original two cards are one up, one down, so you do not know the dealer's total and must play the odds based on the one card showing. You do not need to incorporate this twist into your simulation for this project. Here's what you are required to do:

Run through 12 sets of two decks playing the game. You have an unlimited bankroll (don't you wish!) and bet $2 on each hand. Each time the two decks run out, the hand in play continues with two fresh decks (104 cards). At that point record your standing (plus or minus X dollars). Then start again at 0 for the next deck. So your output will be the 12 results from playing each of the 12 decks, which you can then average or total to determine your overall performance.

What about *your* strategy? That's up to you! But here's the catch—you will assume that you can see neither of the dealer's cards (so you have no idea what cards the dealer has). Choose a strategy to play, and then play it throughout the entire simulation. (Blackjack enthusiasts can consider implementing doubling down and splitting pairs into their simulation, but this is not necessary.)

Provide your instructor with the simulation algorithm, computer code, and output results from each of the 12 decks.

2. *Darts*—Construct and perform a Monte Carlo simulation of a darts game. The rules are

Dart board area	Points
Bullseye	50
Yellow ring	25
Blue ring	15
Red ring	10
White ring	5

From the origin (the center of the bullseye), the radius of each ring is as follows:

Ring	Thickness (in.)	Distance to outer ring edge from the origin (in.)
Bullseye	1.0	1.0
Yellow	1.5	2.5
Blue	2.5	5.0
Red	3.0	8.0
White	4.0	12.0

The board has a radius of 1 ft (12 in.).

Make an assumption about the distribution of how the darts hit on the board. Write an algorithm, and code it in the computer language of your choice. Run 1000 simulations to determine the mean score for throwing five darts. Also, determine which ring has the highest expected value (point value times the probability of hitting that ring).

3. *Craps*—Construct and perform a Monte Carlo simulation of the popular casino game of craps. The rules are as follows:

> There are two basic bets in craps, pass and don't pass. In the *pass* bet, you wager that the shooter (the person throwing the dice) will win; in the *don't pass* bet, you wager that the shooter will lose. We will play by the rule that on an initial roll of 12 ("boxcars"), both pass and don't pass bets are losers. Both are even-money bets.
>
> Conduct of the game:
>
> Roll a 7 or 11 on the first roll: Shooter wins (pass bets win and don't pass bets lose).
>
> Roll a 12 on the first roll: Shooter loses (boxcars, pass and don't pass bets lose).
>
> Roll a 2 or 3 on the first roll: Shooter loses (pass bets lose, don't pass bets win).
>
> Roll 4, 5, 6, 8, 9, 10 on the first roll: This becomes the point. The object then becomes to roll the point again before rolling a 7.
>
> The shooter continues to roll the dice until the point or a 7 appears. Pass bettors win if the shooter rolls the point again before rolling a 7. Don't pass bettors win if the shooter rolls a 7 before rolling the point again.

Write an algorithm and code it in the computer language of your choice. Run the simulation to estimate the probability of winning a pass bet and the probability of winning a don't pass bet. Which is the better bet? As the number of trials increases, to what do the probabilities converge?

Mathematical Derby

Entry's name	Odds
Euler's Folly	7–1
Leapin' Leibniz	5–1
Newton Lobell	9–1
Count Cauchy	12–1
Pumped up Poisson	4–1
Loping L'Hôpital	35–1
Steamin' Stokes	15–1
Dancin' Dantzig	4–1

4. *Horse Race*—Construct and perform a Monte Carlo simulation of a horse race. You can be creative and use odds from the newspaper, or simulate the Mathematical Derby with the entries and odds shown on the left.

Construct and perform a Monte Carlo simulation of 1000 horse races. Which horse won the most races? Which horse won the fewest races? Do these

results surprise you? Provide the tallies of how many races each horse won with your output.

5. *Roulette*—In American roulette, there are 38 spaces on the wheel: 0, 00, and 1–36. Half the spaces numbered 1–36 are red and half are black. The two spaces 0 and 00 are green.

   Simulate the playing of 1000 games betting either red or black (which pay even money, 1:1). Bet \$1 on each game and keep track of your earnings. What are the earnings per game betting red/black according to your simulation? What was your longest winning streak? Longest losing streak?

   Simulate 1000 games betting green (pays 17:1, so if you win, you add \$17 to your kitty, and if you lose, you lose \$1). What are your earnings per game betting green according to your simulation? How does it differ from your earnings betting red/black? What was your longest winning streak betting green? Longest losing streak? Which strategy do you recommend using, and why?

6. *The Price Is Right*—On the popular TV game show "The Price Is Right," at the end of each half hour, the three winning contestants face off in the Showcase Showdown. The game consists of spinning a large wheel with 20 spaces on which the pointer can land, numbered from \$0.05 to \$1.00 in 5¢ increments. The contestant who has won the least amount of money at this point in the show spins first, followed by the one who has won the next most, followed by the biggest winner for that half hour.

   The objective of the game is to obtain as close to \$1.00 as possible without going over that amount with a maximum of two spins. Naturally, if the first player does not go over, the other two will use one or both spins in an attempt to overtake the leader.

   However, what of the person spinning first? If he or she is an expected value decision maker, how high a value on the first spin does he or she need to not want a second spin? Remember, the person can lose if

   (a) Either of the other two players surpasses the player's total or

   (b) The player spins again and goes over \$1.

7. *Let's Make a Deal*—You are dressed to kill in your favorite costume and the host picks you out of the audience. You are offered the choice of three wallets. Two wallets contain a single \$50 bill, and the third contains a \$1000 bill. You choose one of the wallets, 1, 2, or 3. The host, who knows which wallet contains the \$1000, then shows you one of the other two wallets, which has \$50 inside. The host does this purposely because he has at least one wallet with \$50 inside. If he also has the \$1000 wallet, he simply shows you the \$50 one he holds. Otherwise, he just shows you one of his two \$50 wallets. The host then asks you if you want to trade your choice for the one he's still holding. Should you trade?

   Develop an algorithm and construct a computer simulation to support your answer.

5.4 Inventory Model: Gasoline and Consumer Demand

In the previous section, we began modeling probabilistic behaviors using Monte Carlo simulation. In this section we will learn a method to approximate more probabilistic processes. Additionally, we will check to determine how well the simulation duplicates the process under examination. We begin by considering an inventory control problem.

You have been hired as a consultant by a chain of gasoline stations to determine how often and how much gasoline should be delivered to the various stations. Each time gasoline is delivered, a cost of d dollars is incurred, which is in addition to the cost of gasoline and is independent of the amount delivered. The gasoline stations are near interstate highways, so demand is fairly constant. Other factors determining the costs include the capital tied up in the inventory, the amortization costs of equipment, insurance, taxes, and security measures. We assume that, in the short run, the demand and price of gasoline are constant for each station, yielding a constant total revenue as long as the station does not run out of gasoline. Because total profit is total revenue minus total cost, and total revenue is constant by assumption, total profit can be maximized by minimizing total cost. Thus, we identify the following problem: *Minimize the average daily cost of delivering and storing sufficient gasoline at each station to meet consumer demand.*

After discussing the relative importance of the various factors determining the average daily cost, we develop the following model:

$$\text{average daily cost} = f(\text{storage costs, delivery costs, demand rate})$$

Turning our attention to the various submodels, we argue that although the cost of storage may vary with the amount stored, it is reasonable to assume the cost per unit stored would be constant over the range of values under consideration. Similarly, the delivery cost is assumed constant per delivery, independent of the amount delivered, over the range of values under consideration. Plotting the daily demand for gasoline at a particular station is very likely to give a graph similar to the one shown in Figure 5.4a. If the frequency of each demand level over a fixed

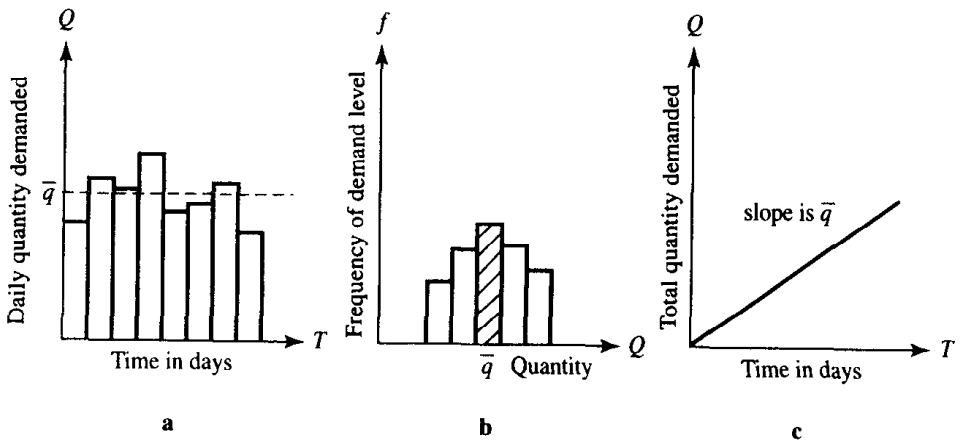


Figure 5.4
A constant demand rate

time period (e.g., 1 year) is plotted, then a plot similar to that shown in Figure 5.4b might be obtained.

If demands are tightly packed around the most frequently occurring demand, then we would accept the daily demand as being constant. In some cases, it may be reasonable to assume a constant demand. Finally, even though the demands occur in discrete time periods, a continuous submodel for demand can be used for simplicity. A continuous submodel is depicted in Figure 5.4c, where the slope of the line represents the constant daily demand. Notice the importance of each of the preceding assumptions in producing the linear submodel.

From these assumptions we will construct in Chapter 12 an analytic model for the average daily cost and use it to compute an optimal time between deliveries and an optimal delivery quantity:

$$T^* = \sqrt{\frac{2d}{sr}}$$

$$Q^* = rT^*$$

where

T^* = optimal time between deliveries in days

Q^* = optimal delivery quantity of gasoline in gallons

r = demand rate in gallons per day

d = delivery cost in dollars per delivery

s = storage cost per gallon per day

We will see how the analytic model depends heavily on a set of conditions that, although reasonable in some cases, would never be met precisely in the real world. It is difficult to develop analytic models that take into account the probabilistic nature of the submodels.

Suppose we decide to check our submodel for constant demand rate by inspecting the sales for the past 1000 days at a particular station. Thus, the data displayed in Table 5.6 are collected.

For each of the 10 intervals of demand levels given in Table 5.6, compute the relative frequency of occurrence by dividing the number of occurrences by the total number of days, 1000. This computation results in an estimate of the probability of occurrence for each demand level. These probabilities are displayed in Table 5.7 and plotted in histogram form in Figure 5.5.

If we are satisfied with the assumption of a constant demand rate, we might estimate this rate at 1550 gallons per day (from Figure 5.5). Then the analytic model could be used to compute the optimal time between deliveries and the delivery quantities from the delivery and storage costs.

Suppose, however, that we are not satisfied with the assumption of constant daily demand. How could we simulate the submodel for the demand suggested by Figure 5.5? First, we could build a cumulative histogram by consecutively adding

Table 5.6 History of demand at a particular gasoline station

Number of gallons demanded	Number of occurrences (in days)
1000–1099	10
1100–1199	20
1200–1299	50
1300–1399	120
1400–1499	200
1500–1599	270
1600–1699	180
1700–1799	80
1800–1899	40
1900–1999	30
	1000

Table 5.7 Probability of the occurrence of each demand level

Number of gallons demanded	Probability of occurrence
1000–1099	0.01
1100–1199	0.02
1200–1299	0.05
1300–1399	0.12
1400–1499	0.20
1500–1599	0.27
1600–1699	0.18
1700–1799	0.08
1800–1899	0.04
1900–1999	0.03
	1.00

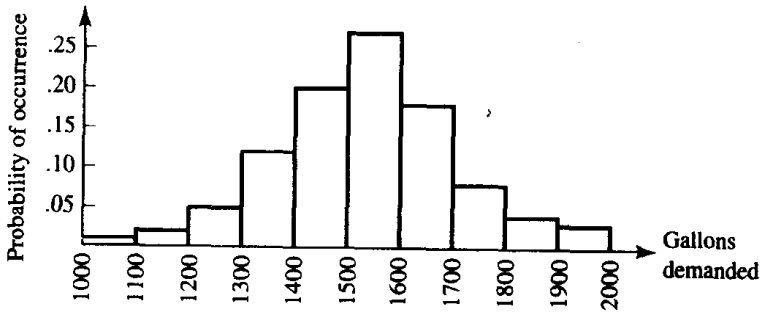


Figure 5.5
The relative frequency of each of the demand intervals in Table 5.7

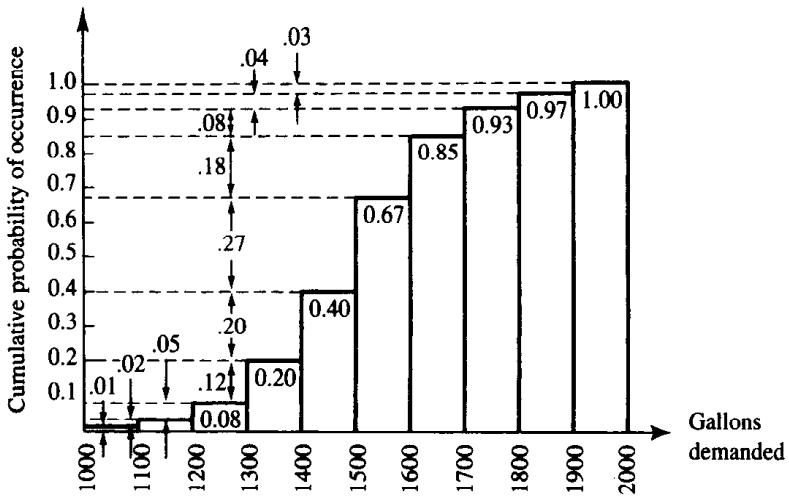


Figure 5.6
A cumulative histogram of the demand submodel from the data in Table 5.7

together the probabilities of each individual demand level, as displayed in Figure 5.6. Note in Figure 5.6 that the difference in height between adjacent columns represents the probability of occurrence of the subsequent demand interval. Thus, we can construct a correspondence between the numbers in the interval $0 \leq x \leq 1$ and the relative occurrence of the various demand intervals. This correspondence is displayed in Table 5.8.

Table 5.8 Using random numbers uniformly distributed over $0 \leq x \leq 1$ to duplicate the occurrence of the various demand intervals

Random number	Corresponding demand	Percent occurrence
$0 \leq x < 0.01$	1000–1099	0.01
$0.01 \leq x < 0.03$	1100–1199	0.02
$0.03 \leq x < 0.08$	1200–1299	0.05
$0.08 \leq x < 0.20$	1300–1399	0.12
$0.20 \leq x < 0.40$	1400–1499	0.20
$0.40 \leq x < 0.67$	1500–1599	0.27
$0.67 \leq x < 0.85$	1600–1699	0.18
$0.85 \leq x < 0.93$	1700–1799	0.08
$0.93 \leq x < 0.97$	1800–1899	0.04
$0.97 \leq x \leq 1.00$	1900–1999	0.03

Thus, if the numbers between 0 and 1 are randomly generated so that each number has an equal probability of occurring, the histogram of Figure 5.5 can be approximated. Using a random number generator on a handheld programmable calculator, we generated random numbers between 0 and 1 and then used the assignment procedure suggested by Table 5.7 to determine the demand interval corresponding to each random number. The results for 1000 and 10,000 trials are presented in Table 5.9.

Table 5.9 A Monte Carlo approximation of the demand submodel

Interval	Number of occurrences/expected no. of occurrences 1000 trials	10,000 trials
1000–1099	8/10	91/100
1100–1199	16/20	198/200
1200–1299	46/50	487/500
1300–1399	118/120	1205/1200
1400–1499	194/200	2008/2000
1500–1599	275/270	2681/2700
1600–1699	187/180	1812/1800
1700–1799	83/80	857/800
1800–1899	34/40	377/400
1900–1999	39/30	284/300
	1000/1000	10,000/10,000

For the gasoline inventory problem, we ultimately want to be able to determine a specific demand, rather than a demand interval, for each day simulated. How can this be accomplished? There are several alternatives. Consider the plot of the midpoints of each demand interval as displayed in Figure 5.7. Because we want a continuous model capturing the trend of the plotted data, we can use methods discussed in Chapter 4.

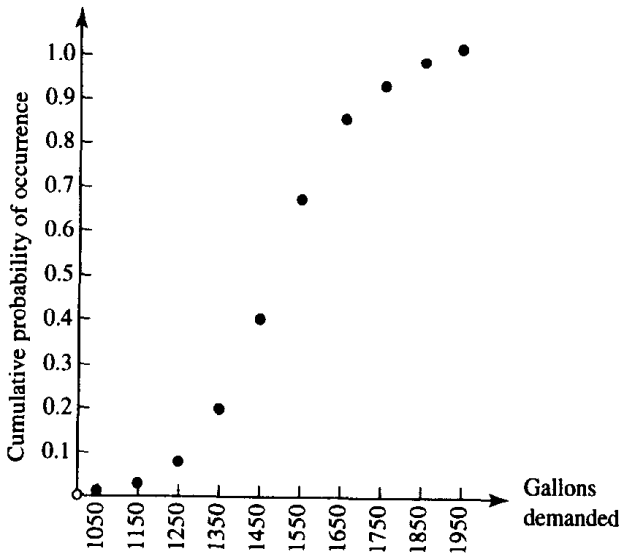


Figure 5.7
A cumulative plot of the demand submodel displaying only the center point of each interval

In many instances, especially where the subintervals are small and the data fairly approximate, a linear spline model is suitable. A linear spline model for the data displayed in Figure 5.7 is presented in Figure 5.8, and the individual spline functions are given in Table 5.10. The interior spline functions—$S_2(q)$–$S_9(q)$—

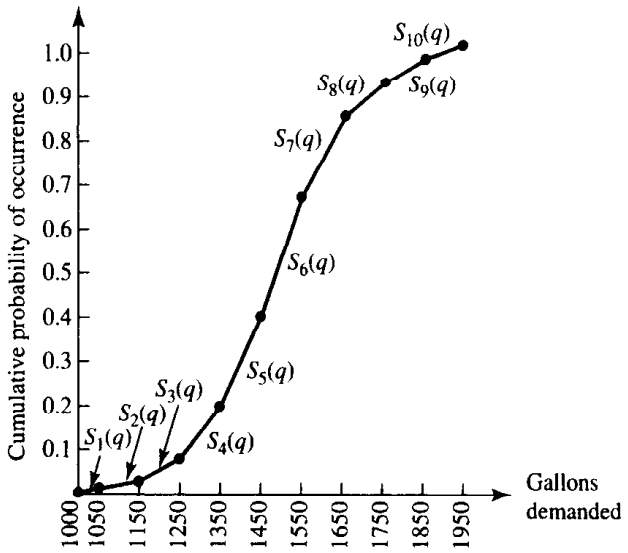


Figure 5.8
A linear spline model for the demand submodel

Table 5.10 Linear splines for the empirical demand submodel

Demand interval	Linear spline
$1000 \le q < 1050$	$S_1(q) = 0.0002q - 0.2$
$1050 \le q < 1150$	$S_2(q) = 0.0002q - 0.2$
$1150 \le q < 1250$	$S_3(q) = 0.0005q - 0.545$
$1250 \le q < 1350$	$S_4(q) = 0.0012q - 1.42$
$1350 \le q < 1450$	$S_5(q) = 0.002q - 2.5$
$1450 \le q < 1550$	$S_6(q) = 0.0027q - 3.515$
$1550 \le q < 1650$	$S_7(q) = 0.0018q - 2.12$
$1650 \le q < 1750$	$S_8(q) = 0.0008q - 0.47$
$1750 \le q < 1850$	$S_9(q) = 0.0004q + 0.23$
$1850 \le q \le 2000$	$S_{10}(q) = 0.0002q + 0.6$

were computed by passing a line through the adjacent data points. $S_1(q)$ was computed by passing a line through (1000, 0) and the first data point (1050, 0.01). $S_{10}(q)$ was computed by passing a line through the points (1850, 0.97) and (2000, 1.00). Note that if we use the midpoints of the intervals, we have to make a decision on how to construct the two exterior splines. If the intervals are small, it is usually easy to construct a linear spline function that captures the trend of the data.

Now suppose we wish to simulate a daily demand for a given day. To do this, we generate a random number x between 0 and 1 and compute a corresponding demand, q. That is, x is the independent variable from which a unique corresponding q is calculated. This calculation is possible because the function depicted in Figure 5.8 is strictly increasing. (Think about whether this situation will always be the case.) Thus, the problem is to find the inverse functions for the splines listed in Table 5.10. For instance, given $x = S_1(q) = 0.0002q - 0.2$, we can solve for $q = (x + 0.2)5000$. In the case of linear splines, it is very easy to find the inverse functions summarized in Table 5.11.

Let's illustrate how Table 5.11 can be used to represent the daily demand submodel. To simulate a demand for a given day, we generate a random number between 0 and 1, say $x = 0.214$. Because $0.20 \leq 0.214 < 0.40$, the spline $q = (x + 2.5)500$ is used to compute $q = 1357$. Thus, 1357 gallons is the simulated demand for that day.

Table 5.11 Inverse linear splines provide for the daily demand as a function of a random number in [0, 1]

Random number	Inverse linear spline
$0 \leq x < 0.01$	$q = (x + 0.2)5000$
$0.01 \leq x < 0.03$	$q = (x + 0.2)5000$
$0.03 \leq x < 0.08$	$q = (x + 0.545)2000$
$0.08 \leq x < 0.20$	$q = (x + 1.42)833.3\underline{3}$
$0.20 \leq x < 0.40$	$q = (x + 2.5)500$
$0.40 \leq x < 0.67$	$q = (x + 3.515)370.37$
$0.67 \leq x < 0.85$	$q = (x + 2.12)555.5\underline{5}$
$0.85 \leq x < 0.93$	$q = (x + 0.47)1250$
$0.93 \leq x < 0.97$	$q = (x - 0.23)2500$
$0.97 \leq x \leq 1.00$	$q = (x - 0.6)5000$

Note that the inverse splines presented in Table 5.11 could have been constructed directly from the data in Figure 5.6 by choosing x as the independent variable instead of q. We will follow this procedure later when computing the cubic spline demand submodel. (The preceding development was presented to help you understand the process and also because it mimics what you will do after studying probability.) Figure 5.6 is an example of a cumulative distribution function. Many types of behavior approximate well-known probability distributions, which can be used as the basis for Figure 5.6 rather than experimental data. The inverse function must then be found to use as the demand submodel in the simulation, and this may prove to be difficult. In such cases, the inverse function is approximated with an empirical model, such as a linear spline or cubic spline. For an excellent introduction to some types of behavior that follow well-known probability distributions, see UMAP 340, "The Poisson Random Process," by Carroll Wilde, listed in the projects at the end of this section.

If we want a smooth continuous submodel for demand, we can construct a cubic spline submodel. We will construct the splines directly as a function of the random number x. That is, using a computer program, we calculate the cubic splines for the following data points:

x	0	0.01	0.03	0.08	0.2	0.4	0.67	0.85	0.93	0.97	1.0
q	1000	1050	1150	1250	1350	1450	1550	1650	1750	1850	2000

The splines are presented in Table 5.12. If the random number $x = 0.214$ is generated, the empirical cubic spline model yields the demand $q = 1350 + 715.5(0.014) - 1572.5(0.014)^2 + 2476(0.014)^3 = 1359.7$ gal.

Table 5.12 An empirical cubic spline model for demand

Random number	Cubic spline
$0 \le x < 0.01$	$S_1(x) = 1000 + 4924.92x + 750788.75x^3$
$0.01 \le x < 0.03$	$S_2(x) = 1050 + 5150.18(x - 0.01) + 22523.66(x - 0.01)^2 - 1501630.8(x - 0.01)^3$
$0.03 \le x < 0.08$	$S_3(x) = 1150 + 4249.17(x - 0.03) - 67574.14(x - 0.03)^2 + 451815.88(x - 0.03)^3$
$0.08 \le x < 0.20$	$S_4(x) = 1250 + 880.37(x - 0.08) + 198.24(x - 0.08)^2 - 4918.74(x - 0.08)^3$
$0.20 \le x < 0.40$	$S_5(x) = 1350 + 715.46(x - 0.20) - 1572.51(x - 0.20)^2 + 2475.98(x - 0.20)^3$
$0.40 \le x < 0.67$	$S_6(x) = 1450 + 383.58(x - 0.40) - 86.92(x - 0.40)^2 + 140.80(x - 0.40)^3$
$0.67 \le x < 0.85$	$S_7(x) = 1550 + 367.43(x - 0.67) + 27.12(x - 0.67)^2 + 5655.69(x - 0.67)^3$
$0.85 \le x < 0.93$	$S_8(x) = 1650 + 926.92(x - 0.85) + 3081.19(x - 0.85)^2 + 11965.43(x - 0.85)^3$
$0.93 \le x < 0.97$	$S_9(x) = 1750 + 1649.66(x - 0.93) + 5952.90(x - 0.93)^2 + 382645.25(x - 0.93)^3$
$0.97 \le x \le 1.00$	$S_{10}(x) = 1850 + 3962.58(x - 0.97) + 51870.29(x - 0.97)^2 - 576334.88(x - 0.97)^3$

An empirical submodel for demand can be constructed in a variety of other ways. For example, rather than using the intervals for gallons demanded as given in Table 5.6, we can use smaller intervals. If the intervals are small enough, the midpoint of an interval could be a reasonable approximation to the demand for the entire interval. Thus, a cumulative histogram similar to that in Figure 5.6 could serve as a submodel directly. If preferred, a continuous submodel could be constructed readily from the refined data.

The purpose of our discussion has been to demonstrate how a submodel for a probabilistic behavior can be constructed using Monte Carlo simulation and experimental data. Now let's see how the inventory problem can be simulated in general terms.

An inventory strategy consists of specifying a delivery quantity Q and a time T between deliveries, given values for storage cost per gallon per day s and a delivery cost d. If s and d are known, then a specific inventory strategy can be tested using a Monte Carlo simulation algorithm, as follows:

Summary of Monte Carlo Inventory Algorithm Terms

- Q Delivery quantity of gasoline in gallons
- T Time between deliveries in days
- I Current inventory in gallons
- d Delivery cost in dollars per delivery
- s Storage cost per gallon per day
- C Total running cost
- c Average daily cost
- N Number of days to run the simulation

K Days remaining in the simulation

x_i A random number in the interval [0, 1]

q_i A daily demand

Flag An indicator used to terminate the algorithm

Monte Carlo Inventory Algorithm

Input Q, T, d, s, N

Output c

Step 1 Initialize:

$$K = N$$
$$I = 0$$
$$C = 0$$
$$\text{Flag} = 0$$

Step 2 Begin the next inventory cycle with a delivery:

$$I = I + Q$$
$$C = C + d$$

Step 3 Determine if the simulation will terminate during this cycle:

If $T \geq K$, then set $T = K$ and Flag = 1

Step 4 Simulate each day in the inventory cycle (or portion remaining):

For $i = 1, 2, \ldots, T$, do Steps 5–9

Step 5 Generate the random number x_i.

Step 6 Compute q_i using the demand submodel.

Step 7 Update the current inventory: $I = I - q_i$.

Step 8 Compute the daily storage cost and total running cost, unless the inventory has been depleted: If $I \leq 0$, then set $I = 0$ and GOTO Step 9.
Else $C = C + I * s$.

Step 9 Decrement the number of days remaining in the simulation:

$$K = K - 1$$

Step 10 If Flag = 0, then GOTO Step 2. Else GOTO Step 11.

Step 11 Compute the average daily cost: $c = C/N$.

Step 12 Output c.
STOP

Various strategies can now be tested with the algorithm to determine the average daily costs. You probably want to refine the algorithm to keep track of other measures of effectiveness, such as unsatisfied demands and number of days without gasoline, as suggested in the following problem set.

5.4 Problems

1. Modify the inventory algorithm to keep track of unfilled demands and the total number of days that the gasoline station is without gasoline for at least part of the day.

2. Most gasoline stations have a storage capacity Q_{max} that cannot be exceeded. Refine the inventory algorithm to take this consideration into account. Because of the probabilistic nature of the demand submodel at the end of the inventory cycle, there might still be significant amounts of gasoline remaining. If several cycles occur in succession, the excess might build up to Q_{max}. Because there is a financial cost in carrying excess inventory, this situation would be undesirable. What alternatives can you suggest? Modify the inventory algorithm to take your alternatives into account.

3. In many situations, the time T between deliveries and the order quantity Q is not fixed. Instead, an order is placed for a specific amount of gasoline. Depending on how many orders are placed in a given time interval, the time to fill an order varies. You have no reason to believe that the performance of the delivery operation will change. Therefore, you have examined records for the past 100 deliveries and found the following lag times or extra days required to fill your order:

Lag time (in days)	Number of occurrences
2	10
3	25
4	30
5	20
6	13
7	2
Total:	100

   Construct a Monte Carlo simulation for the lag time submodel. If you have a handheld calculator or computer available, test your submodel by running 1000 trials and comparing the number of occurrences of the various lag times with the historical data.

4. Problem 3 suggests an alternative inventory strategy. When the inventory reaches a certain level (an order point), an order can be placed for an optimal amount of gasoline. Construct an algorithm that simulates this process and incorporates probabilistic submodels for demand and lag times. How could you use this algorithm to search for the optimal order point and the optimal order quantity?

5. In the case in which a gasoline station runs out of gas, the customer is simply going to go to another station. In many situations (name a few), however,

some customers will place a back order or collect a rain check. If the order is not filled within a time period varying from customer to customer in a probabilistic fashion, the customer will cancel his or her order. Suppose we examine historical data for 1000 customers and find the data shown in Table 5.13. That is, 200 customers will not even place an order, and an additional 150 customers will cancel if the order is not filled within 1 day.

Table 5.13 Hypothetical data for a back order submodel

Number of days customer is willing to wait before canceling	Number of occurrences	Cumulative occurrences
0	200	200
1	150	350
2	200	550
3	200	750
4	150	900
5	50	950
6	50	1000
	1000	

(a) Construct a Monte Carlo simulation for the back order submodel. If you have a calculator or computer available, test your submodel by running 1000 trials and comparing the number of occurrences of the various cancellations with the historical data.

(b) Consider the algorithm you modified in Problem 1. Further modify the algorithm to consider back orders. Do you think back orders should be penalized in some fashion? If so, how would you do it?

5.4 Projects

1. Complete the requirements of UMAP module 340, "The Poisson Random Process," by Carroll O. Wilde. Probability distributions are introduced to obtain practical information on random arrival patterns, interarrival times or gaps between arrivals, waiting line buildup, and service loss rates. The Poisson distribution, the exponential distribution, and Erlang's formulas are used. The module requires an introductory probability course, the ability to use summation notation, and basic concepts of the derivative and the integral from calculus. Prepare a 10-min summary of the module for a classroom presentation.

2. Assume a storage cost of \$0.001 per gallon per day and a delivery charge of \$500 per delivery. Construct a computer code of the algorithm you constructed in Problem 4 and compare various order points and order quantity strategies.

5.5 Queuing Models

EXAMPLE 1 A Harbor System

Consider a small harbor with unloading facilities for ships. Only one ship can be unloaded at any one time. Ships arrive for unloading of cargo at the harbor, and the time between the arrival of successive ships varies from 15 to 145 min. The unloading time required for a ship depends on the type and amount of cargo and varies from 45 to 90 min. We seek answers to the following questions:

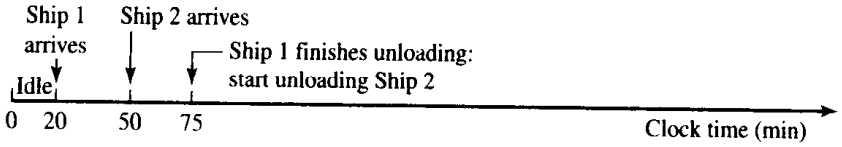
1. What is the average and maximum times per ship in the harbor?
2. If the *waiting time* for a ship is the time between its arrival and the start of unloading, what are the average and maximum waiting times per ship?
3. What percentage of the time are the unloading facilities idle?
4. What is the length of the longest queue?

To obtain some reasonable answers, we can simulate the activity in the harbor using a computer or programmable calculator. We assume the arrival times between successive ships and unloading time per ship are uniformly distributed over their respective time intervals. For instance, the arrival time between ships can be any integer between 15 and 145, and any integer within that interval can appear with equal likelihood. Before giving a general algorithm to simulate the harbor system, let's consider a hypothetical situation with five ships.

We have the following data for each ship:

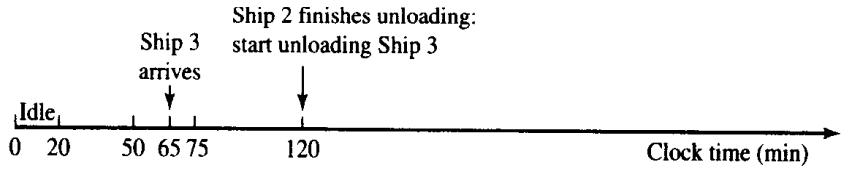
	Ship 1	Ship 2	Ship 3	Ship 4	Ship 5
Time between successive ships	20	30	15	120	25
Unload time	55	45	60	75	80

Because Ship 1 arrives 20 min after the clock commences at $t = 0$ min, the harbor facilities are idle for 20 min at the start. Ship 1 immediately begins to unload. The unloading takes 55 min; meanwhile, Ship 2 arrives on the scene at $t = 20 + 30 = 50$ min after the clock begins. Ship 2 cannot start to unload until Ship 1 finishes unloading at $t = 20 + 55 = 75$ min. This means that Ship 2 must wait $75 - 50 = 25$ min before unloading begins. The situation is depicted in the following timeline diagram:



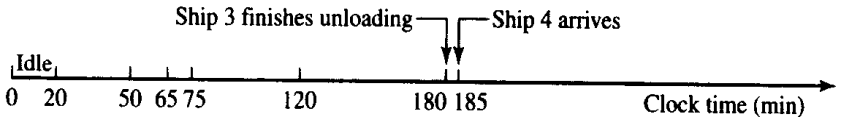
Timeline 1

Now before Ship 2 starts to unload, Ship 3 arrives at time $t = 50 + 15 = 65$ min. Because the unloading of Ship 2 starts at $t = 75$ min and it takes 45 min to unload, unloading Ship 3 cannot start until $t = 75 + 45 = 120$ min, when Ship 2 is finished. Thus, Ship 3 must wait $120 - 65 = 55$ min. The situation is depicted in the next timeline diagram:



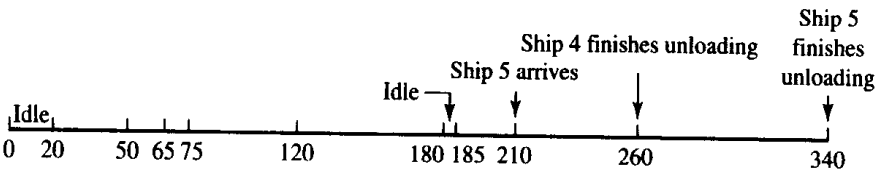
Timeline 2

Ship 4 does not arrive in the harbor until $t = 65 + 120 = 185$ min. Therefore, Ship 3 has already finished unloading at $t = 120 + 60 = 180$ min, and the harbor facilities are idle for $185 - 180 = 5$ min. Moreover, the unloading of Ship 4 commences immediately upon its arrival, as depicted in the next diagram:



Timeline 3

Finally, Ship 5 arrives at $t = 185 + 25 = 210$ min, before Ship 4 finishes unloading at $t = 185 + 75 = 260$ min. Thus, Ship 5 must wait $260 - 210 = 50$ min before it starts to unload. The simulation is complete when Ship 5 finishes unloading at $t = 260 + 80 = 340$ min. The final situation is shown in the next diagram:



Timeline 4

In Figure 5.9, we summarize the waiting and unloading times for each of the five hypothetical ship arrivals. In Table 5.14, we summarize the results of the entire simulation of the five hypothetical ships. Note the total waiting time spent by all five ships before unloading is 130 min. This waiting time represents a cost to the shipowners and is a source of customer dissatisfaction with the docking facilities. On the other hand, the docking facility has only 25 min of total idle time. It is in use 315 out of the total 340 min in the simulation, or approximately 93% of the time.

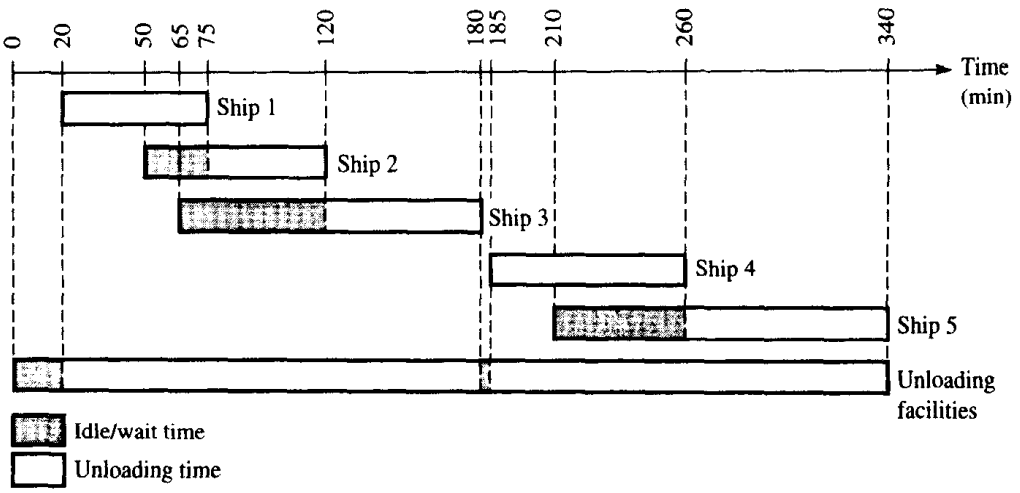


Figure 5.9
Idle and unloading times for the ships and docking facilities

Table 5.14 Summary of the harbor system simulation

Ship no.	Random time between ship arrivals	Arrival time	Start service	Queue length at arrival	Wait time	Random unload time	Time in harbor	Dock idle time
1	20	20	20	0	0	55	55	20
2	30	50	75	1	25	45	70	0
3	15	65	120	2	55	60	115	0
4	120	185	185	0	0	75	75	5
5	25	210	260	1	50	80	130	0
Totals (if appropriate):					130			25
Averages (if appropriate):					26	63	89	

Note: All times are given in minutes after the start of the clock at time $t = 0$.

Suppose the owners of the docking facilities are concerned with the quality of service they are providing and want various management alternatives to be evaluated to determine if improvement in service justifies the added cost. Several statistics can help in evaluating the quality of the service. For example, the maximum time a ship spends in the harbor is 130 min by Ship 5, whereas the average is 89 min (Table 5.14). Generally, customers are very sensitive to the amount of time spent waiting. In this example, the maximum amount of time spent waiting for a facility is 55 min, whereas the average time spent waiting is 26 min. Some customers are apt to take their business elsewhere if queues are too long. In this case, the longest queue is two. The following Monte Carlo simulation algorithm computes such statistics to assess various management alternatives.

Summary of Harbor System Algorithm Terms

between$_i$ Time between successive arrivals of Ships i and $i - 1$ (a random integer varying between 15 and 145 min)

arrive$_i$ Time from start of clock at $t = 0$ when Ship i arrives at the harbor for unloading

unload$_i$ Time required to unload Ship i at the dock (a random integer varying between 45 and 90 min)

start$_i$ Time from start of clock at which Ship i commences its unloading

idle$_i$ Time for which dock facilities are idle immediately *before* commencement of unloading Ship i

wait$_i$ Time Ship i waits in the harbor after arrival before unloading commences

finish$_i$ Time from start of clock at which service for Ship i is completed at the unloading facilities

harbor$_i$ Total time Ship i spends in the harbor

HARTIME Average time per ship in the harbor

MAXHAR Maximum time of a ship in the harbor

WAITIME Average waiting time per ship before unloading

MAXWAIT Maximum waiting time of a ship

IDLETIME Percentage of total simulation time unloading facilities are idle

Harbor System Simulation Algorithm

Input Total number n of ships for the simulation.

Output HARTIME, MAXHAR, WAITIME, MAXWAIT, and IDLETIME.

Step 1 Randomly generate between$_1$ and unload$_1$. Then set arrive$_1$ = between$_1$.

Step 2 Initialize all output values:

$$\text{HARTIME} = \text{unload}_1, \quad \text{MAXHAR} = \text{unload}_1,$$
$$\text{WAITIME} = 0, \quad \text{MAXWAIT} = 0, \quad \text{IDLETIME} = \text{arrive}_1$$

Step 3 Calculate finish time for unloading of Ship$_1$:

$$\text{finish}_1 = \text{arrive}_1 + \text{unload}_1$$

Step 4 For $i = 2, 3, \ldots, n$, do Steps 5–16.

Step 5 Generate the random pair of integers between$_i$ and unload$_i$ over their respective time intervals.

Step 6 Assuming the time clock begins at $t = 0$ min, calculate the time of arrival for Ship$_i$:

$$\text{arrive}_i = \text{arrive}_{i-1} + \text{between}_i$$

Step 7 Calculate the time difference between the arrival of Ship$_i$ and the finish time for unloading the previous Ship$_{i-1}$:

$$\text{timediff} = \text{arrive}_i - \text{finish}_{i-1}$$

Step 8 For nonnegative timediff, the unloading facilities are idle:

$$\text{idle}_i = \text{timediff} \quad \text{and} \quad \text{wait}_i = 0$$

For negative timediff, Ship$_i$ must wait before it can unload:

$$\text{wait}_i = -\text{timediff} \quad \text{and} \quad \text{idle}_i = 0$$

Step 9 Calculate the start time for unloading Ship$_i$:

$$\text{start}_i = \text{arrive}_i + \text{wait}_i$$

Step 10 Calculate the finish time for unloading Ship_i:

$$\text{finish}_i = \text{start}_i + \text{unload}_i$$

Step 11 Calculate the time in harbor for Ship_i:

$$\text{harbor}_i = \text{wait}_i + \text{unload}_i$$

Step 12 Sum harbor_i into total harbor time HARTIME for averaging.

Step 13 If $\text{harbor}_i > \text{MAXHAR}$, then set $\text{MAXHAR} = \text{harbor}_i$. Otherwise leave MAXHAR as is.

Step 14 Sum wait_i into total waiting time WAITIME for averaging.

Step 15 Sum idle_i into total idle time IDLETIME.

Step 16 If $\text{wait}_i > \text{MAXWAIT}$, then set $\text{MAXWAIT} = \text{wait}_i$. Otherwise leave MAXWAIT as is.

Step 17 Set $\text{HARTIME} = \text{HARTIME}/n$, $\text{WAITIME} = \text{WAITIME}/n$, and $\text{IDLETIME} = \text{IDLETIME}/\text{finish}_n$.

Step 18 OUTPUT (HARTIME, MAXHAR, WAITIME, MAXWAIT, IDLETIME)
STOP

Table 5.15 gives the results, according to the preceding algorithm, of six independent simulation runs of 100 ships each.

Table 5.15 Harbor system simulation results for 100 ships

Average time of a ship in the harbor	106	85	101	116	112	94
Maximum time of a ship in the harbor	287	180	233	280	234	264
Average waiting time of a ship	39	20	35	50	44	27
Maximum waiting time of a ship	213	118	172	203	167	184
Percentage of time dock facilities are idle	0.18	0.17	0.15	0.20	0.14	0.21

Note: All times are given in minutes. Time between successive ships is 15–145 min. Unloading time per ship varies from 45 to 90 min.

Now suppose you are a consultant for the owners of the docking facilities. What would be the effect of hiring additional labor or acquiring better equipment for unloading cargo so that the unloading time interval is reduced to between 35 and 75 min per ship? Table 5.16 gives the results based on our simulation algorithm.

Table 5.16 Harbor system simulation results for 100 ships

Average time of a ship in the harbor	74	62	64	67	67	73
Maximum time of a ship in the harbor	161	116	167	178	173	190
Average waiting time of a ship	19	6	10	12	12	16
Maximum waiting time of a ship	102	58	102	110	104	131
Percentage of time dock facilities are idle	0.25	0.33	0.32	0.30	0.31	0.27

Note: All times are given in minutes. Time between successive ships is 15–145 min. Unloading time per ship varies from 35 to 75 min.

You can see from Table 5.16 that a reduction of the unloading time per ship by 10 to 15 min decreases the time ships spend in the harbor, especially the waiting times. However, the percentage of the total time during which the dock facilities are idle nearly doubles. The situation is favorable for shipowners because it increases the availability of each ship for hauling cargo over the long run. Thus, the traffic coming into the harbor is likely to increase. If the traffic increases to the extent that the time between successive ships is reduced to between 10 and 120 min, the simulated results are shown in Table 5.17. We can see from this table that the ships again spend more time in the harbor with the increased traffic, but now harbor facilities are idle much less of the time. Moreover, both the shipowners and the dock owners are benefiting from the increased business.

Table 5.17 Harbor system simulation results for 100 ships

Average time of a ship in the harbor	114	79	96	88	126	115
Maximum time of a ship in the harbor	248	224	205	171	371	223
Average waiting time of a ship	57	24	41	35	71	61
Maximum waiting time of a ship	175	152	155	122	309	173
Percentage of time dock facilities are idle	0.15	0.19	0.12	0.14	0.17	0.06

Note: All times are given in minutes. Time between successive ships is 10–120 min. Unloading time per ship varies from 35 to 75 min.

Suppose now that we are not satisfied with the assumption that the arrival time between ships (i.e., their interarrival times) and the unloading time per ship are uniformly distributed over the time intervals $15 \leq \text{between}_i \leq 145$ and $45 \leq \text{unload}_i \leq 90$, respectively. We decide to collect experimental data for the harbor

Table 5.18 Data collected for 1200 ships using the harbor facilities

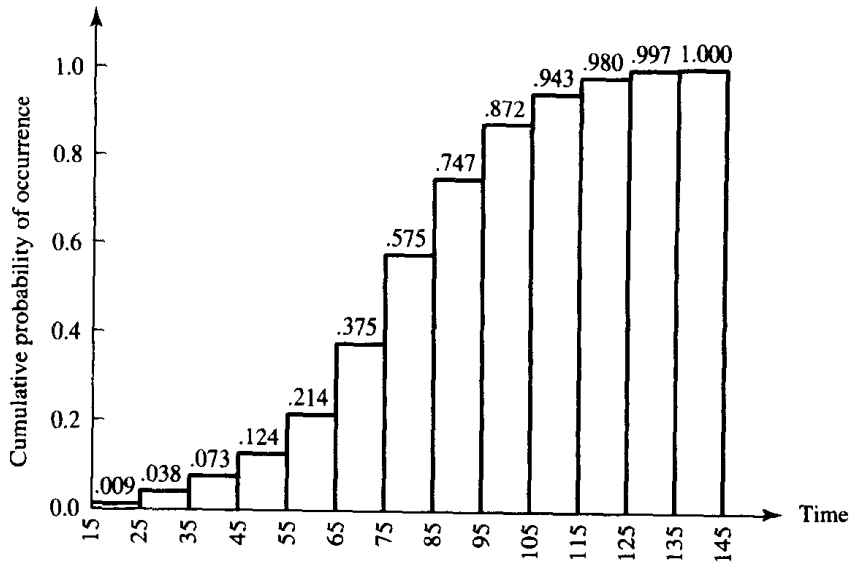
Time between arrivals	Number of occurrences	Probability of occurrence	Unloading time	Number of occurrences	Probability of occurrence
15–24	11	0.009			
25–34	35	0.029			
35–44	42	0.035	45–49	20	0.017
45–54	61	0.051	50–54	54	0.045
55–64	108	0.090	55–59	114	0.095
65–74	193	0.161	60–64	103	0.086
75–84	240	0.200	65–69	156	0.130
85–94	207	0.172	70–74	223	0.185
95–104	150	0.125	75–79	250	0.208
105–114	85	0.071	80–84	171	0.143
115–124	44	0.037	85–90	109	0.091
125–134	21	0.017		1200	1.000
135–145	3	0.003			
	1200	1.000			

Note: All times are given in minutes.

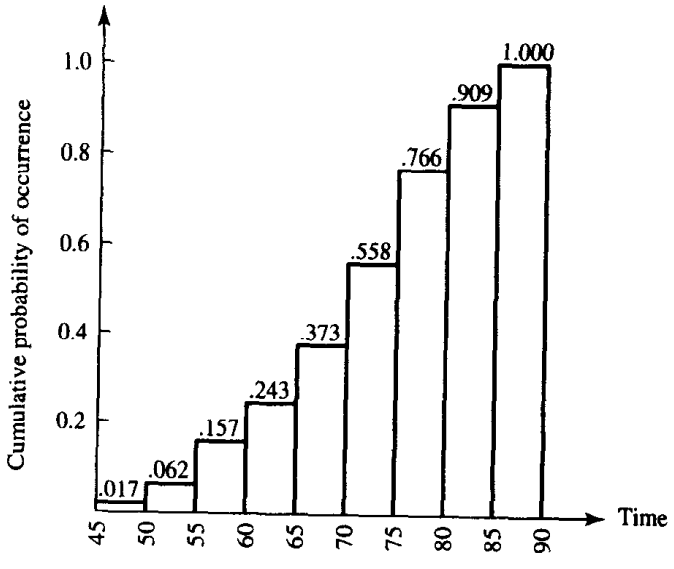
system and incorporate the results in our model, as discussed for the demand submodel in the previous section. We observe (hypothetically) 1200 ships using the harbor to unload their cargoes, and we collect the data displayed in Table 5.18.

Following the procedures outlined in Section 5.4, we consecutively add together the probabilities of each individual time interval between arrivals as well as probabilities of each individual unloading time interval. These computations result in cumulative histograms depicted in Figure 5.10.

Figure 5.10
Cumulative histograms of the time between ship arrivals and the unloading times, from the data in Table 5.18



a. Time between arrivals



b. Unloading time

Next we use random numbers uniformly distributed over the interval $0 \leq x \leq 1$ to duplicate the various interarrival times and unloading times based on the cumulative histograms. We then use the midpoints of each interval and construct linear splines through adjacent data points. (We ask you to complete this construction in Problem 1.) Because it is easy to calculate the inverse splines directly, we do so and summarize the results in Tables 5.19 and 5.20.

Table 5.19 Linear segment submodels provide for the time between arrivals of successive ships as a function of a random number in the interval [0, 1]

Random number interval	Corresponding arrival time	Inverse linear spline
$0 \leq x < 0.009$	$15 \leq b < 20$	$b = 555.6x + 15.0000$
$0.009 \leq x < 0.038$	$20 \leq b < 30$	$b = 344.8x + 16.8966$
$0.038 \leq x < 0.073$	$30 \leq b < 40$	$b = 285.7x + 19.1429$
$0.073 \leq x < 0.124$	$40 \leq b < 50$	$b = 196.1x + 25.6863$
$0.124 \leq x < 0.214$	$50 \leq b < 60$	$b = 111.1x + 36.2222$
$0.214 \leq x < 0.375$	$60 \leq b < 70$	$b = 62.1x + 46.7080$
$0.375 \leq x < 0.575$	$70 \leq b < 80$	$b = 50.0x + 51.2500$
$0.575 \leq x < 0.747$	$80 \leq b < 90$	$b = 58.1x + 46.5698$
$0.747 \leq x < 0.872$	$90 \leq b < 100$	$b = 80.0x + 30.2400$
$0.872 \leq x < 0.943$	$100 \leq b < 110$	$b = 140.8x - 22.8169$
$0.943 \leq x < 0.980$	$110 \leq b < 120$	$b = 270.3x - 144.8649$
$0.980 \leq x < 0.997$	$120 \leq b < 130$	$b = 588.2x - 456.4706$
$0.997 \leq x \leq 1.000$	$130 \leq b \leq 145$	$b = 5000.0x - 4855$

Table 5.20 Linear segment submodels provide for the unloading time of a ship as a function of a random number in the interval [0, 1]

Random number interval	Corresponding unloading time	Inverse linear spline
$0 \leq x < 0.017$	$45 \leq u < 47.5$	$u = 147x + 45.000$
$0.017 \leq x < 0.062$	$47.5 \leq u < 52.5$	$u = 111x + 45.611$
$0.062 \leq x < 0.157$	$52.5 \leq u < 57.5$	$u = 53x + 49.237$
$0.157 \leq x < 0.243$	$57.5 \leq u < 62.5$	$u = 58x + 48.372$
$0.243 \leq x < 0.373$	$62.5 \leq u < 67.5$	$u = 38.46x + 53.154$
$0.373 \leq x < 0.558$	$67.5 \leq u < 72.5$	$u = 27x + 57.419$
$0.558 \leq x < 0.766$	$72.5 \leq u < 77.5$	$u = 24x + 59.087$
$0.766 \leq x < 0.909$	$77.5 \leq u < 82.5$	$u = 35x + 50.717$
$0.909 \leq x \leq 1.000$	$82.5 \leq u \leq 90$	$u = 82.41x + 7.582$

Finally, we incorporate our linear spline submodels into the simulation model for the harbor system by generating between_i and unload_i for $i = 1, 2, \ldots, n$ in Steps 1 and 5 of our algorithm, according to the rules displayed in Tables 5.19 and

5.20. Employing these submodels, Table 5.21 gives the results of six independent simulation runs of 100 ships each. ■

Table 5.21 Harbor system simulation results for 100 ships

Average time of a ship in the harbor	108	95	125	78	123	101
Maximum time of a ship in the harbor	237	188	218	133	250	191
Average waiting time of a ship	38	25	54	9	53	31
Maximum waiting time of a ship	156	118	137	65	167	124
Percentage of time dock facilities are idle	0.09	0.09	0.08	0.12	0.06	0.10

Note: Based on the data exhibited in Table 5.18. All times are given in minutes.

EXAMPLE 2 Morning Rush Hour

In the previous example, we initially considered a harbor system with a single facility for unloading ships. Such problems are often called *single-server queues.* In this example, we consider a system with four elevators, illustrating *multiple-server queues.* We discuss the problem and present the algorithm in Appendix B.

Consider an office building with 12 floors in a metropolitan area of some city. During the morning rush hour, from 7:50 to 9:10 A.M., workers enter the lobby of the building and take an elevator to their floor. There are four elevators servicing the building. The time between arrivals of the customers at the building varies in a probabilistic manner every 0–30 sec, and upon arrival each customer selects the first available elevator (numbered 1–4). When a person enters an elevator and selects the floor of destination, the elevator waits 15 sec before closing its doors. If another person arrives within the 15-sec interval, the waiting cycle is repeated. If no person arrives within the 15-sec interval, the elevator departs to deliver all of its passengers. We assume no other passengers are picked up along the way. After delivering its last passenger, the elevator returns to the main floor, picking up no passengers on the way down. The maximum occupancy of an elevator is 12 passengers. When a person arrives in the lobby and no elevator is available (because all four elevators are transporting their load of passengers), a queue begins to form in the lobby.

The management of the building wants to provide good elevator service to its customers and is interested in exactly what service it is now giving. Some customers claim that they have to wait too long in the lobby before an elevator returns. Others complain that they spend too much time riding the elevator, and still others say that there is considerable congestion in the lobby during the morning rush hour. What is the real situation? Can the management resolve these complaints by a more effective means of scheduling or utilizing the elevators?

We wish to simulate the elevator system using an algorithm for computer implementation that will give answers to the following questions:

1. How many customers are actually being serviced in a typical morning rush hour?

2. If the *waiting time* of a person is the time the person stands in a queue—the time from arrival at the lobby until entry into an available elevator—what are the average and maximum times a person waits in a queue?
3. What is the length of the longest queue? (The answer to this question will provide the management with information about congestion in the lobby.)
4. If the *delivery time* is the time it takes a customer to reach his or her floor after arrival in the lobby, including any waiting time for an available elevator, what are the average and maximum delivery times?
5. What are the average and maximum times a customer actually spends in the elevator?
6. How many stops are made by each elevator? What percentage of the total morning rush hour time is each elevator actually in use?

An algorithm is presented in Appendix B. ■

5.5 Problems

1. Using the data from Table 5.18 and the cumulative histograms of Figure 5.10, construct cumulative plots of the time between arrivals and unloading time submodels (as in Figure 5.7). Calculate equations for the linear splines over each random number interval. Compare your results with the inverse splines given in Tables 5.19 and 5.20.
2. Use a smooth polynomial to fit the data in Table 5.18 to obtain arrivals and unloading times. Compare results to those in Tables 5.19 and 5.20.
3. Modify the ship harbor system algorithm to keep track of the number of ships waiting in the queue.
4. Most small harbors have a maximum number of ships N_{max} that can be accommodated in the harbor area while they wait to be unloaded. If a ship cannot get into the harbor, assume it goes elsewhere to unload its cargo. Refine the ship harbor algorithm to take these considerations into account.
5. Suppose the owners of the docking facilities decide to construct a second facility to accommodate the unloading of more ships. When a ship enters the harbor, it goes to the next available facility, which is facility 1 if both facilities are available. Using the same assumption for interarrival times between successive ships and unloading times as in the initial text example, modify the algorithm for a system with two facilities.
6. Construct a Monte Carlo simulation of a baseball game. Use individual batting statistics to simulate the probability of a single, double, triple, home run, or an out. In a more refined model, how would you handle walks, hit batsman, steals, and double plays?

5.5 Projects

1. Write a computer simulation to implement the ship harbor algorithm.

2. Write a computer simulation to implement a baseball game between your two favorite teams (see Problem 6).

3. Pick a traffic intersection with a traffic light. Collect data on vehicle arrival times and clearing times. Build a Monte Carlo simulation to model traffic flow at this intersection.

4. In the Los Angeles County School District, substitute teachers are placed in a pool and paid whether they teach or not. It is assumed that if the need for substitutes exceeds the size of the pool, classes can be covered by regular teachers, but at a higher pay rate. Letting x represent the number of substitutes needed on a given day, S the pool size, p the amount of pay for pool members, and r the daily overtime rate, we have for the cost

$$C(x, S) = \begin{cases} pS & \text{if } x < S \\ pS + (x - S)r & \text{if } x \geq S \end{cases}$$

Here, we assume $p < r$.

(a) Use the data provided for the number of substitutes needed on Mondays to simulate the situation in an attempt to optimize the pool size. The optimized pool will be the one with the lowest expected cost to the school district. Use for pay rates, $p = \$45$ and $r = \$81$. Assume that the data are distributed uniformly.

i. Make 500 simulations at each value of S from $S = 100$ to $S = 900$ in steps of 100 using the averages of the 500 runs to estimate the cost for each value of S.

Demand for substitute teachers on Mondays

Number of teachers	Relative percentage	Cumulative percentage
201–275	2.7	2.7
276–350	2.7	5.4
351–425	2.7	8.1
426–500	2.7	10.8
501–575	16.2	27
576–650	10.8	37.8
651–725	48.6	86.4
726–800	8.1	94.5
801–875	2.7	97.2
876–950	2.7	99.9

ii. Narrow the search for the best value of S to an interval of length 200 and make runs of 1000 simulations for each of ten equally spaced values of S in this interval.

iii. Continue the process narrowing the search for the optimal size of the pool, each time stepping the values of S a smaller amount and increasing the number of iterations for better accuracy. When you have determined the optimal value for the pool size, S, submit your choice with substantiating evidence.

(b) Redo part (a) with $p = \$36$ and $r = \$81$.

(c) Redo part (a) using the data provided for Tuesdays. Assume $p = \$45$ and $r = \$81$.

(d) Redo part (c) using the data for Tuesdays assuming $p = \$36$ and $r = \$81$.

Demand for substitute teachers on Tuesdays

Number of teachers	Relative percentage	Cumulative percentage
201–275	2.5	2.5
276–350	2.5	5.0
351–425	5.0	10.0
426–500	7.5	17.5
501–575	12.5	30.0
576–650	17.5	47.5
651–725	42.5	90.0
726–800	5.0	95.0
801–875	2.5	97.5
876–950	2.5	100.0