2.2.5 Write $\gamma(x) = \alpha(x) - \beta(x)$, where $\alpha(x) = a(x \bmod q)$ and $\beta(x) = r\lfloor x/q \rfloor$, with $m = aq + r$ and $r = m \bmod a$. Prove that, if $r < q$, then for all $x \in \mathcal{X}_m$, both $\alpha(x)$ and $\beta(x)$ are in $\{0, 1, 2, \ldots, m-1\}$.

2.2.6 Is Algorithm 2.2.1 valid if m is not prime? If not, how should it be modified?

2.2.7 (*a*) Implement a correct version of `Random` that uses floating-point arithmetic and do a timing study. (*b*) Comment.

2.2.8 Prove that, if a, x, q are positive integers, then $ax \bmod aq = a(x \bmod q)$.

2.2.9 You have been hired as a consultant by *XYZ Inc.* to assess the market potential of a relatively inexpensive *hardware* random-number generator they might develop for simulation-computing applications. List all the technical reasons you can think of to convince them this is a bad idea.

2.2.10 There are exactly 400 points in each of the figures in Example 2.2.7. (*a*) Why? (*b*) How many points would there be if a were not a full-period multiplier?

2.2.11 Let m be the largest prime modulus less than or equal to $2^{15} - 1$. (See Exercise 2.1.6.) (*a*) Compute all the corresponding modulus-compatible full-period multipliers. (*b*) Comment on how this result relates to random-number generation on systems that support 16-bit integer arithmetic only.

2.2.12 (*a*) Prove that, if m is prime with $m \bmod 4 = 1$, then a is a full-period multiplier if and only if $m - a$ is also a full-period multiplier. (*b*) What if $m \bmod 4 = 3$?

2.2.13 If $m = 2^{31} - 1$, compute the $x \in \mathcal{X}_m$ for which $7^x \bmod m = 48\,271$.

2.2.14 The lines on the scatterplot in Figure 2.2.4 associated with the multiplier $a = 16807$ appear to be vertical. Argue that the lines must *not* be vertical, using the fact that $(a, m) = (16807, 2^{31} - 1)$ is a full-period generator.

2.2.15 Figure out whether the multipliers associated with $m = 2^{31} - 1$ given by Fishman (2001) are modulus-compatible: $a = 630\,360\,016$, $a = 742\,938\,285$, $a = 950\,706\,376$, $a = 1\,226\,874\,159$, $a = 62\,089\,911$, and $a = 1\,343\,714\,438$.

2.3 MONTE CARLO SIMULATION

In this section, we will consider Monte Carlo simulation. (See the taxonomy in Figure 1.1.1.) Specifically, the discussion will focus on the estimation of one or more probabilities by using the functions in the library `rng` to implement an experimental technique whose validity is based on what is known as the *frequency* theory of probability.

2.3.1 Probability

There are two approaches to probability, both of which will be illustrated in this section. One approach is experimental (empirical), the other is theoretical (axiomatic).

Empirical Probability

Definition 2.3.1 If a random experiment is repeated n times and if n_a is the number of times event $\mathcal{A}$ occurs ($n_a \leq n$), then the *relative frequency* of occurrence of event $\mathcal{A}$ is n_a/n. The *frequency theory of probability* asserts that this relative frequency converges to the probability of $\mathcal{A}$ as $n \to \infty$

$$\Pr(\mathcal{A}) = \lim_{n \to \infty} \frac{n_a}{n}.$$

Based as it is on the acquisition of experimental data, Definition 2.3.1 is also called the *empirical* or *experimental* definition of probability. By its reliance on arbitrarily many replications of the random experiment, the equation in Definition 2.3.1 is related to the strong and weak *laws of large numbers*. Although the frequency theory of probability has limited theoretical importance, it is the cornerstone of experimental statistics and simulation. Monte Carlo simulation uses the frequency theory of probability in a natural way. The idea is simple yet profound. A computational model is built that uses a random-number generator to simulate the random experiment. The simulated experiment is then repeated many times (n) and the number of times event $\mathcal{A}$ occurs (n_a) is recorded. If n is large, the ratio n_a/n is a good *point estimate* of the probability $\Pr(\mathcal{A})$.

Axiomatic Probability. In contrast to the frequency theory, the formal study of probability is usually based on the *axiomatic* theory of probability. This is the familiar set-theoretic approach to probability, in which the primary emphasis is on the mathematical construction of a sample space with an associated probability function $\Pr(\mathcal{A})$ defined for all events $\mathcal{A}$ in the sample space. The axiomatic theory of probability and the frequency theory of probability are best viewed as complementary. Doing one brings insight to the other. The best solution to any probability problem is a mathematical solution established via the axiomatic method and verified experimentally via an independent Monte Carlo simulation. Many probability problems, however, are just too hard to solve mathematically. For those problems, Monte Carlo simulation may be the only viable approach. The axiomatic and frequency theories of probability are also complementary in the use of the latter to explain the former—that is, the significance of a mathematically derived probability $\Pr(\mathcal{A})$ is commonly explained by interpreting the probability as the relative frequency with which the event $\mathcal{A}$ would occur if the random experiment were to be repeated many times.

Example 2.3.1

Roll two dice and observe the up faces. This classic random experiment is simple enough that it can be analyzed by using a special case of the axiomatic approach: Construct a finite sample space with all points equally likely, then to evaluate $\Pr(\mathcal{A})$, count the points in $\mathcal{A}$ and divide by the cardinality of the sample space. That is, think of the dice as distinguishable (say, one is green and the other is red) and construct the sample space as a set of $6^2 = 36$ ordered pairs of the form (a, b), where both a and b can take on any integer value between 1 and 6:

$$\begin{array}{cccccc}
(1,1) & (1,2) & (1,3) & (1,4) & (1,5) & (1,6) \\
(2,1) & (2,2) & (2,3) & (2,4) & (2,5) & (2,6) \\
(3,1) & (3,2) & (3,3) & (3,4) & (3,5) & (3,6) \\
(4,1) & (4,2) & (4,3) & (4,4) & (4,5) & (4,6) \\
(5,1) & (5,2) & (5,3) & (5,4) & (5,5) & (5,6) \\
(6,1) & (6,2) & (6,3) & (6,4) & (6,5) & (6,6).
\end{array}$$

If the dice are fair, the 36 points in the sample space are equally likely, each with probability 1/36. If the two up faces are summed, an integer-valued random variable, say X, is defined, having as possible values 2 through 12, inclusive. The probability associated with each possible value of the sum can be found by counting points in the sample space. For example, there are six points in the sample space corresponding to

the sum 7 (this is A). Thus, if two dice are rolled, the probability that the up faces will sum to 7 is $\Pr(X = 7) = 6/36 = 1/6$. In this way, the following table of probabilities can be constructed:

sum, x	2	3	4	5	6	7	8	9	10	11	12
$\Pr(X = x)$	1/36	2/36	3/36	4/36	5/36	6/36	5/36	4/36	3/36	2/36	1/36

As a consistency check, note that the probabilities are nonnegative and sum to 1.

Interpretation. Because the (axiomatic) probability of rolling a 7 is 1/6, in 6000 replications of the two-dice experiment, we expect to see a total of 7 occur approximately 1000 times. This is the relative-frequency interpretation of the mathematically derived (axiomatic) probability 1/6; in the "long run," 7 will occur 1/6 of the time. In the "short run," the relative frequency of 7's can be quite different from 1/6, a fact that is well known to any gambler.

Alternatively, if it were not so easy to figure out the probability of rolling a 7 via the axiomatic approach—for example, if the dice were not fair—then $\Pr(X = 7)$ could be estimated by replicating the experiment many times and calculating the relative frequency of occurrence of 7's. Doing this with a random-number generator is an excellent simple example of what Monte Carlo simulation is all about. (See Exercise 2.3.4.)

Probability estimation via Monte Carlo simulation will be illustrated in this section with two examples of classic probability problems. In both cases, the axiomatic solution and Monte Carlo estimates are compared. Before we do so, however, a few important definitions are needed.

2.3.2 Random Variates

Definition 2.3.2 A random *variate* is an algorithmically generated realization of a random *variable*.*

If Random is a good random-number generator, a *Uniform*(0, 1) random variate is what is generated by the assignment `u = Random()`.

Uniform Random Variates. Given the ability to generate a *Uniform*(0, 1) random variate, what if we need a random variate, say x, that is *Uniform*(a, b)? That is, suppose a and b are real-valued parameters with $a < b$ and that we want to generate values of x in such a way that all real numbers between a and b are equally likely. How should this be done?

The answer is to first use `u = Random()` to generate a *Uniform*(0, 1) random variate u, and to then transform from u to x via the equation $x = a + (b - a)u$, as shown in Figure 2.3.1. Values of u between 0.0 and 1.0 are mapped linearly to values of x between a and b, as illustrated by this set of equivalences:

$$\begin{aligned} 0 < u < 1 &\iff 0 < (b-a)u < b-a \\ &\iff a < a + (b-a)u < a + (b-a) \\ &\iff a < x < b. \end{aligned}$$

*See Chapters 6 and 7 for a discussion of discrete and continuous random variables.

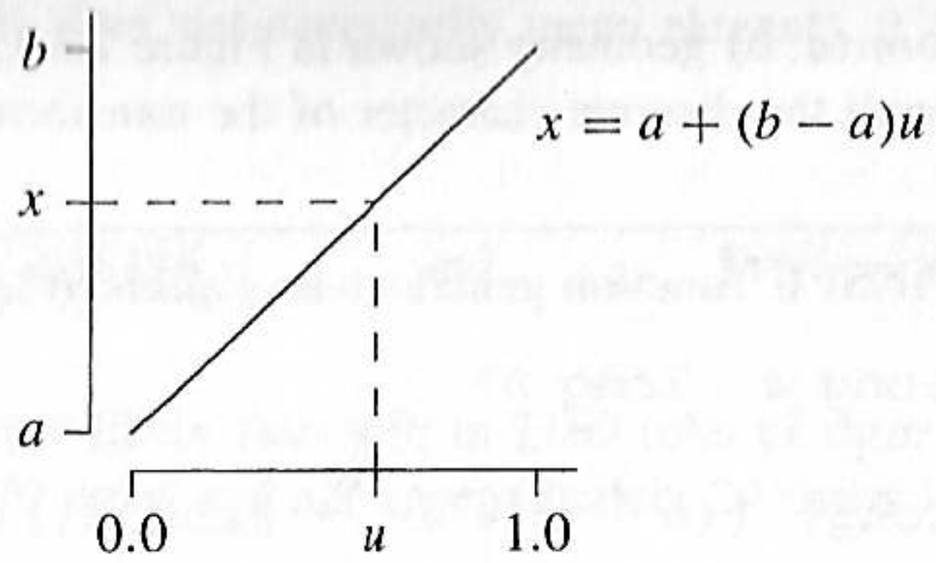

Figure 2.3.1 Geometry associated with *Uniform*(a, b) variate generation.

Definition 2.3.3 This ANSI C function generates a $Uniform(a, b)$ random variate.

```
double Uniform(double a, double b)                    // use a < b
{
   return (a + (b - a) * Random());
}
```

Example 2.3.2
A point x is selected at random in the interval (a, b) to form two subintervals of length $x - a$ and $b - x$. What is the probability that the larger subinterval is more than twice the length of the smaller subinterval? For particular values of a, b, the relative-frequency (Monte Carlo) approach to probability is easily applied in this case by generating x as a $Uniform(a, b)$ random variate. The details are left as Exercise 2.3.10.

Equilikely Random Variates. $Uniform(0, 1)$ random variates can also be used to generate integer-valued random variates. In particular, let a and b be integer-valued parameters with $a < b$, and suppose we want to generate values of an integer-valued random variate x in such a way that all integers between a and b inclusive are equally likely. In this case, x is an $Equilikely(a, b)$ random variate. How can this random variate be generated?

If u = `Random()` then x can be generated via the transformation $x = a + \lfloor (b - a + 1)\, u \rfloor$. That is,

$$\begin{aligned} 0 < u < 1 &\iff 0 < (b - a + 1)u < b - a + 1 \\ &\iff 0 \le \lfloor (b - a + 1)u \rfloor \le b - a \\ &\iff a \le a + \lfloor (b - a + 1)u \rfloor \le b \\ &\iff a \le x \le b \end{aligned}$$

and so values of u between 0.0 and 1.0 are mapped linearly in a discrete manner to integer values of x between a and b inclusive. The geometric representation of this transformation is identical

to the corresponding *Uniform*(a, b) geometry shown in Figure 2.3.1, except that the diagonal line will stairstep, in keeping with the discrete character of the transformation.

Definition 2.3.4 This ANSI C function generates an *Equilikely*(a, b) random variate.

```
long Equilikely(long a, long b)                    // use a < b
{
   return (a + (long) ((b - a + 1) * Random()));
}
```

Example 2.3.3
To generate a random variate x that simulates rolling two fair dice and summing the resulting up faces, use

```
x = Equilikely(1, 6) + Equilikely(1, 6);
```

Note that this is *not* equivalent to

```
x = Equilikely(2, 12);
```

Example 2.3.4
To select an element x at random from the array $a[0], a[1], \ldots, a[n-1]$, use

```
i = Equilikely(0, n - 1);  // pick an array index at random
x = a[i];
```

See Section 6.5 for more discussion of this notion.

We are now ready to consider two complete classic Monte Carlo simulation examples. The first makes use of *Equilikely*(a, b) random variates, the other makes use of *Uniform*(a, b) random variates. Many other types of random variates are considered in later chapters, primarily in discrete-event-simulation applications.

2.3.3 Galileo's Dice

Example 2.3.5
Three fair dice are rolled and the random variable X is the sum of the three up faces. Which sum is more likely, a 9 or a 10? This example is alleged to have been solved by Galileo when asked by a gambler to explain why 10's seemed to appear more often than 9's as the sum of three dice. The axiomatic theory of probability can be used to solve this problem in several ways; the most direct, albeit tedious, approach is based on an extension of Example 2.3.1. That is, the sample space for the three-dice random experiment can be constructed as a set of $6^3 = 216$ equally likely possible outcomes. By listing all 216 of these and counting the ones that yield a sum of 9 or 10,

respectively (axiomatic does not necessarily mean elegant), it can be verified that the probabilities are

$$\Pr(X = 9) = \frac{25}{216} \cong 0.116 \qquad \text{and} \qquad \Pr(X = 10) = \frac{27}{216} = 0.125.$$

So, a 10 is slightly more likely than a 9; in 2160 rolls of three dice, we expect to see a 10 approximately 270 times and a 9 approximately 20 times less.

Program `galileo`. As an alternative, Monte Carlo approach to Example 2.3.5, we can use the sum of three *Equilikely*(1, 6) random variates to simulate the rolling of three dice and use the computed relative frequencies of 9's and 10's to estimate these two probabilities. As it turns out, it is no more difficult to compute the relative frequencies of all the other possible sums as well and, in this way, estimate the probability of each possible sum between 3 and 18. The program `galileo` does exactly that.

The simplicity of program `galileo` is compelling—that is the appeal of simulation. The drawback of Monte Carlo simulation, however, is that the $n \to \infty$ limit operation in Definition 2.3.1 can only be approximated; as a result, program `galileo` can only produce probability *estimates*. Moreover, as the following example illustrates, relative-frequency probability estimates can converge slowly and erratically. Indeed, the $n \to \infty$ limit in Definition 2.3.1 is *not* the traditional type of limit studied in calculus. That is, there is no *guarantee* that larger values of n will always produce more accurate probability estimates. There is, however, the *expectation* that this will be so. The mathematics involved in making this statement more precise is beyond the scope of this book.

Given that relative-frequency probability estimates based on a finite number of replications (finite n) have an inherent uncertainty, an important issue remains—how accurate are these estimates? For a probability-estimation application with modest accuracy requirements, generally 1000 to 10 000 replications is sufficient. For example, in Chapter 8, we will see that, to achieve an estimate of the three-dice probability $\Pr(X = 10) = 0.125$ that is accurate to ± 0.01 (with 95% confidence), approximately 4400 replications are required. For applications with more stringent accuracy requirements, even more replications would be required to reduce the uncertainty to an acceptable level.

For now, we will largely avoid the issue of accuracy versus the number of replications, except to observe that personal computers and workstations are inexpensive, time on them is free for many people, and it is reasonable to expect that, as n becomes larger, the accuracy of the probability estimates will tend to improve.

Example 2.3.6

Figure 2.3.2 illustrates the relatively slow and somewhat erratic convergence of relative-frequency probability estimates for the three-dice random experiment. Three sequences of $\Pr(X = 10)$ estimates are shown, corresponding to the three `rng` initial seeds indicated. A point is plotted for $n = 20, 40, \ldots, 1000$. The horizontal line represents the axiomatic probability 0.125. A figure for $\Pr(X = 9)$ estimates would be similar.

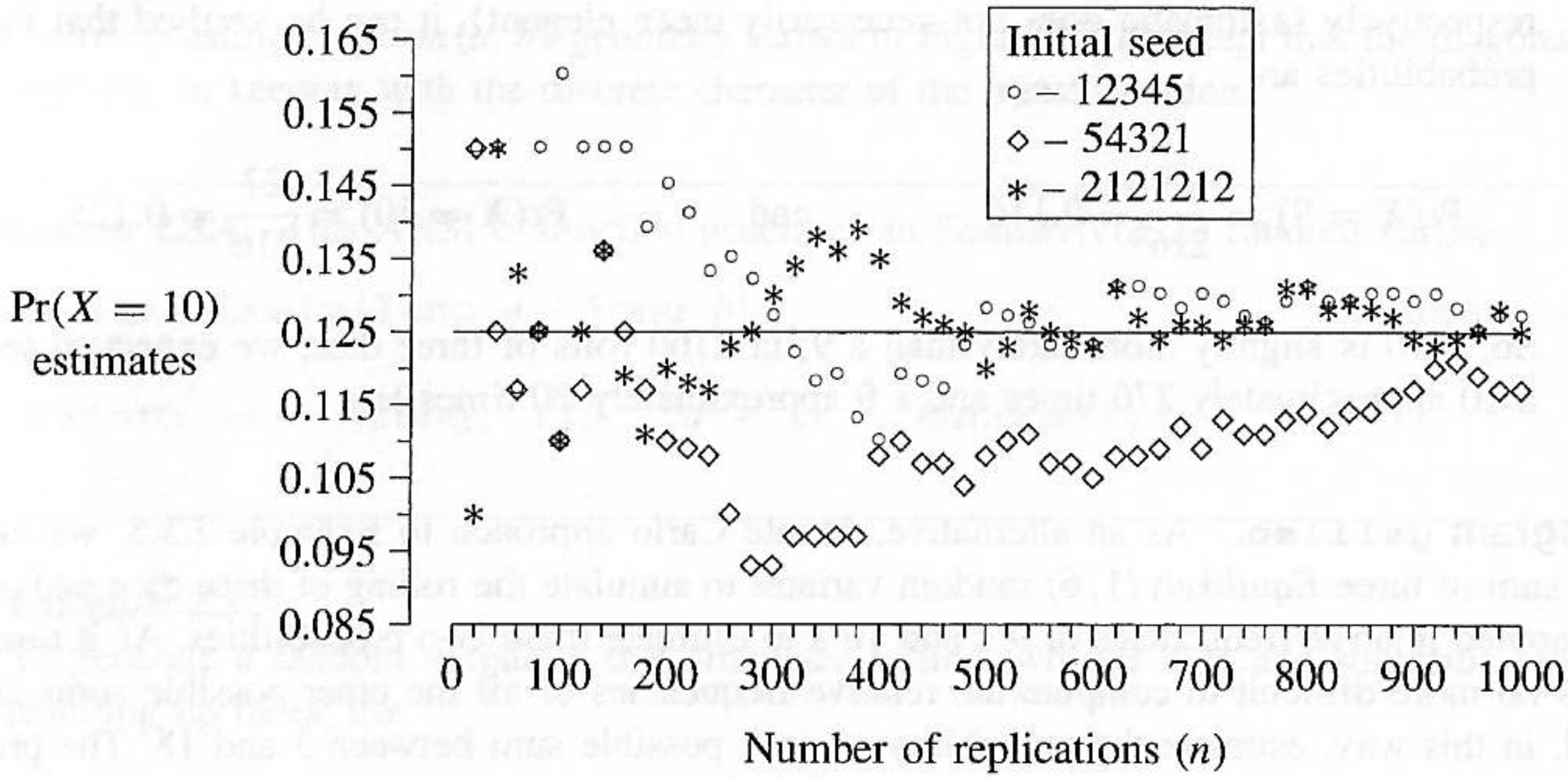

Figure 2.3.2 Program `galileo` output.

Only probability estimates up to the first 1000 replications are shown. Do not interpret this to mean that there is something magic about 1000. There is an inherent uncertainty in any Monte Carlo probability estimate. The magnitude of the uncertainty is dictated, in part, by the number of replications; increasing the number of replications reduces the uncertainty.

Multiple Initial Seeds. Because program `galileo` uses `PutSeed(0)` to initialize the state of the random-number generator, each time the program is run the user is prompted to supply an initial seed. As Example 2.3.6 illustrates, different initial seeds will cause different sequences of three-dice results to be generated and, in that way, different probability estimates will be produced. It is *always* good practice to run a Monte Carlo simulation program multiple times, using a different initial seed for each run, and pay close attention to how the results vary from run to run. If the variability is too large, the number of replications (n) per run should be increased (perhaps a lot—see Chapter 8).

2.3.4 Geometric Applications

The Monte Carlo-simulation solution to the three-dice problem was based on the use of *Equilikely*(a, b) random variates. The use of this discrete random variate is common in Monte Carlo simulation because of the large variety of stochastic models based on a "select *at random* from the finite set $\{a, \ldots, b\}$" characterization. For comparison, we now consider continuous Monte Carlo simulation examples with geometric applications based on the use of *Uniform*(a, b) random variates to simulate the selection of a point "*at random* in the interval (a, b)."

Example 2.3.7

Many Monte Carlo-simulation applications that use *Uniform*(a, b) random variates involve the experimental solution of geometry-based probability problems. To solve these problems, it is typical to use a *Uniform*(a, b) random variate to position points in two or more dimensions at random, subject to geometric constraints.

For example, to generate a point (x, y) at random inside a rectangle with opposite corners at (α_1, β_1) and (α_2, β_2) as illustrated on the left-hand side of Figure 2.3.3, use

```
x = Uniform(α1, α2);
y = Uniform(β1, β2);
```

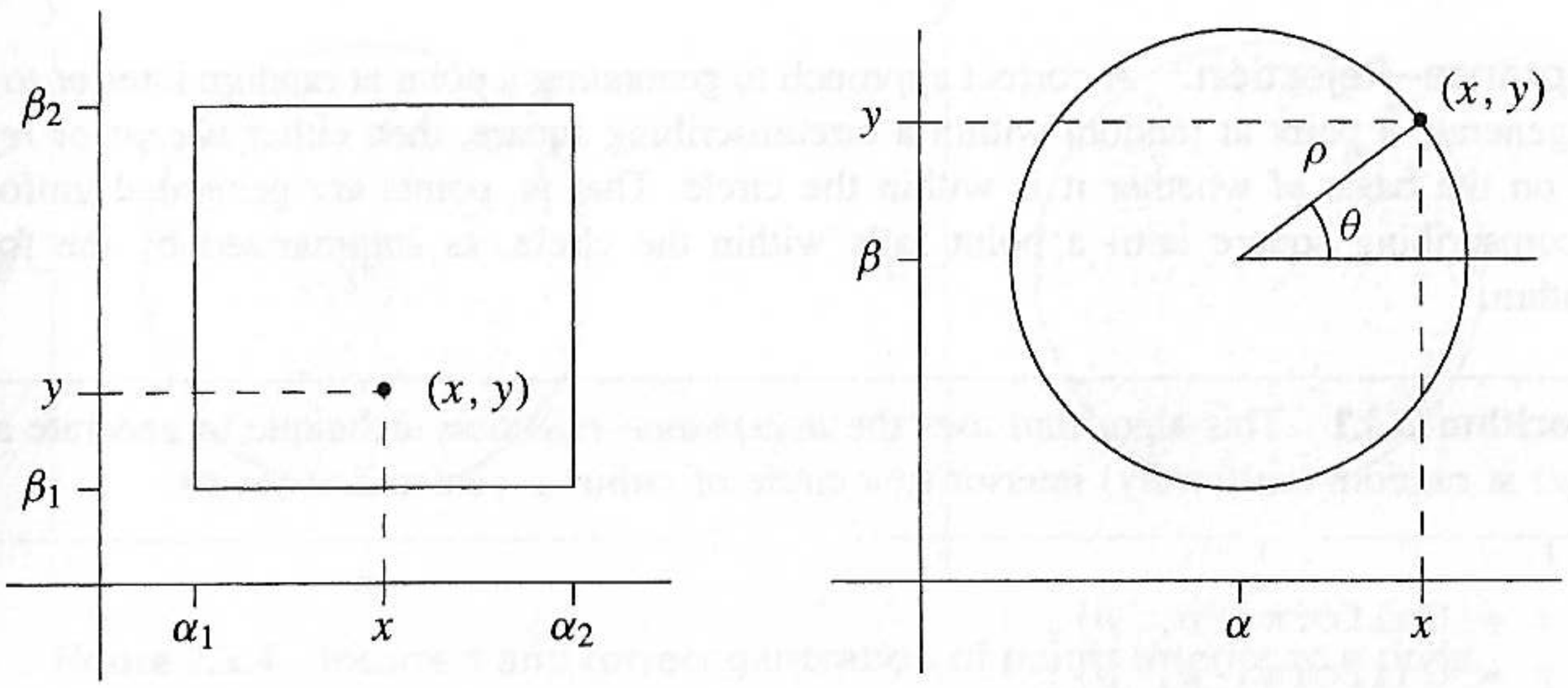

Figure 2.3.3 Generating random points inside a rectangle and on the circumference of a circle.

The bivariate random variate (x, y) so generated is *Uniform* within the rectangle—all locations are equally likely. Similarly, to generate a point (x, y) at random on the *circumference* of a circle with radius ρ and center (α, β), as illustrated on the right-hand side of Figure 2.3.3, use

```
θ = Uniform(-π, π);
x = α + ρ * cos(θ);
y = β + ρ * sin(θ);
```

The resulting bivariate random variate (x, y) is *Uniform* on the circumference of the circle. In both cases the phrase "at random" is a synonym for *Uniform*, properly interpreted relative to the geometry of the figure.

Example 2.3.8
It might seem natural to generalize the second part of Example 2.3.7 to generate a point (x, y) at random *interior* to the circle of radius ρ centered at (α, β) as follows:

```
θ = Uniform(-π, π);
r = Uniform(0, ρ);
x = α + r * cos(θ);
y = β + r * sin(θ);
```

This algorithm is *not* correct, however, because the distribution of points so generated will not be uniform interior to the circle. Instead, points will be more densely distributed close to the center than close to the circumference, as is illustrated in Example 2.3.9.

Acceptance–Rejection. A correct approach to generating a point at random interior to a circle is to generate a point at random within a circumscribing square, then either *accept* or *reject* the point on the basis of whether it is within the circle. That is, points are generated uniformly in a circumscribing square until a point falls within the circle, as summarized by the following algorithm.

Algorithm 2.3.1 This algorithm uses the *acceptance–rejection* technique to generate a point (x, y) at random (uniformly) interior to a circle of radius ρ centered at (α, β).

```
do {
   x = Uniform(-ρ, ρ);
   y = Uniform(-ρ, ρ);
} while (x * x + y * y >= ρ * ρ);                    // rejection
x = α + x;
y = β + y;
return (x, y);
```

The probability of acceptance in the `do-while` loop in Algorithm 2.3.1 is the ratio of the area of the circle to the area of the circumscribed square, which is

$$p = \frac{\pi\rho^2}{4\rho^2} = \frac{\pi}{4} \cong 0.785.$$

Therefore, the algorithm will loop once with probability p, or twice with probability $p(1-p)$, or three times with probability $p(1-p)^2$, etc., so that the expected (average) number of passes through the loop per point generated is

$$p + 2p(1-p) + 3p(1-p)^2 + 4p(1-p)^3 + \cdots.$$

It can be shown that this infinite geometric series converges to $1/p = 4/\pi \cong 1.273$ (see Section 6.1) and so the expected number of passes through the do-while loop in Algorithm 2.3.1 is reasonably small.

Although classic and efficient, Algorithm 2.3.1 is somewhat esthetically unsatisfactory because it "wastes" at least two calls to Random with probability $1 - p = 1 - \pi/4 \cong 0.215$. That is, it seems reasonable to expect that there is a *synchronized* algorithm that generates the (x, y) pairs using *exactly* two calls to Random. Such an algorithm exists; see Exercise 2.3.9.

Example 2.3.9

The scatterplot on the left-hand side of Figure 2.3.4 was generated using the (incorrect) algorithm in Example 2.3.8. The increased density of points near the origin is evident. In contrast, the figure on the right-hand side of Figure 2.3.4 was generated using the (correct) acceptance–rejection technique in Algorithm 2.3.1. (For both figures, the number of (x, y) points is 400, generated with an rng initial seed of 12345.)

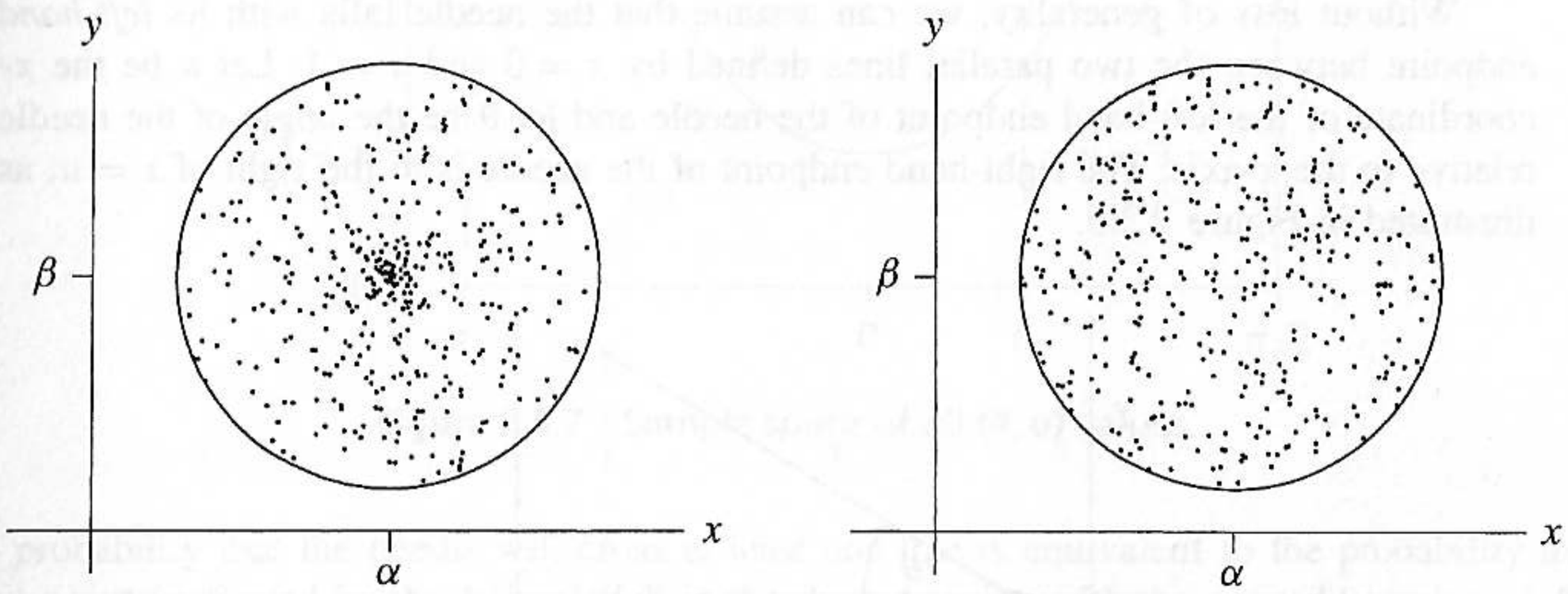

Figure 2.3.4 Incorrect and correct generation of points interior to a circle.

2.3.5 Buffon's Needle

The *Uniform*(a, b) random variate Monte Carlo simulation example in this section is the classic *Buffon's needle problem*, which is one of the oldest known problems in geometric probability. Buffon was a French naturalist of the eighteenth century who first posed and solved the problem that bears his name. The solution of this problem suggests a Monte Carlo technique for estimating π.

Example 2.3.10

Suppose that an infinite family of infinitely long vertical lines are spaced one unit apart in the (x, y) plane. If a needle of length $r > 0$ (and negligible width) is dropped at random onto the plane, what is the probability that it will land crossing at least one line?

One realization of this experiment is shown in Figure 2.3.5. In the particular realization depicted, the needle does not cross one or more of the vertical lines. Obviously, the probability of one or more crossings depends on three quantities: the length of the needle r, the horizontal position of one end of the needle (we arbitrarily choose the left-hand endpoint), and the angle orientation of the needle from horizontal.

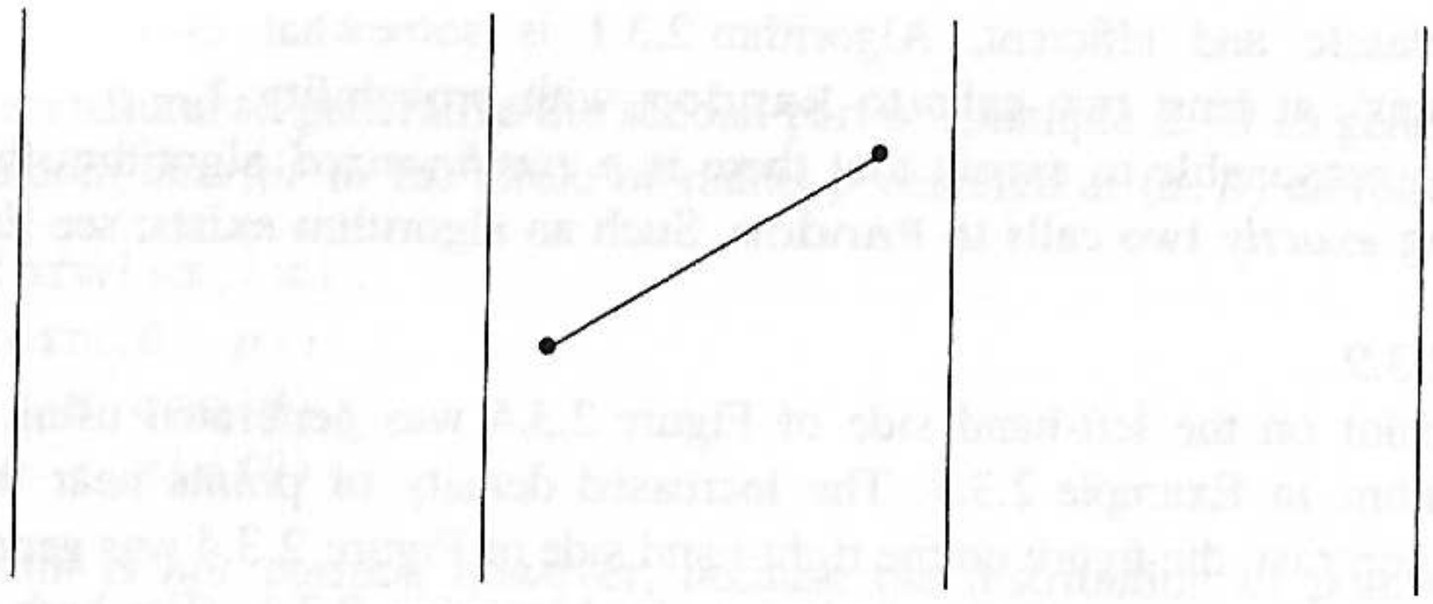

Figure 2.3.5 Buffon needle realization.

Without loss of generality, we can assume that the needle falls with its *left-hand* endpoint between the two parallel lines defined by $x = 0$ and $x = 1$. Let u be the x-coordinate of the left-hand endpoint of the needle and let θ be the angle of the needle relative to the x-axis. The right-hand endpoint of the needle is to the right of $x = u$, as illustrated in Figure 2.3.6.

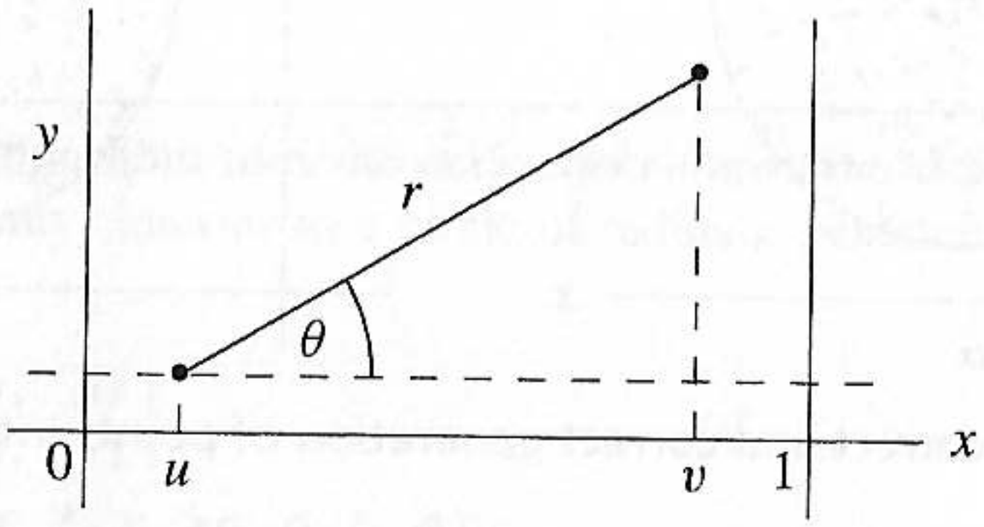

Figure 2.3.6 Buffon needle geometry.

The x-coordinate of the *right-hand* endpoint of the needle is $v = u + r\cos\theta$ and the needle crosses at least one line if and only if $v > 1$. If the phrase "dropped at random" is interpreted (modeled) to mean that u and θ are independent $Uniform(0, 1)$ and $Uniform(-\pi/2, \pi/2)$ random variables respectively, then the dropping of the needle can be simulated by generating two $Uniform(a, b)$ random variates, as in program `buffon`.

Program `buffon`. Program `buffon` is a Monte Carlo simulation that can be used to estimate the probability that the needle will cross at least one line. Note, in particular, the use of

```
PutSeed(-1);                                  // any negative integer will do
GetSeed(&seed);                          // trap the value of the initial seed
   :
printf("with an initial seed of %ld", seed);
```

as the mechanism for initializing the random-number generator. By using this approach, each time the program is run, a different initial seed will be supplied by the system clock and printed along with other program output.

Interestingly, an inspection of program `buffon` illustrates how to solve the problem by using the axiomatic approach. That is, this program generates points (θ, u) at random (uniformly) throughout the rectangle illustrated in Figure 2.3.7.

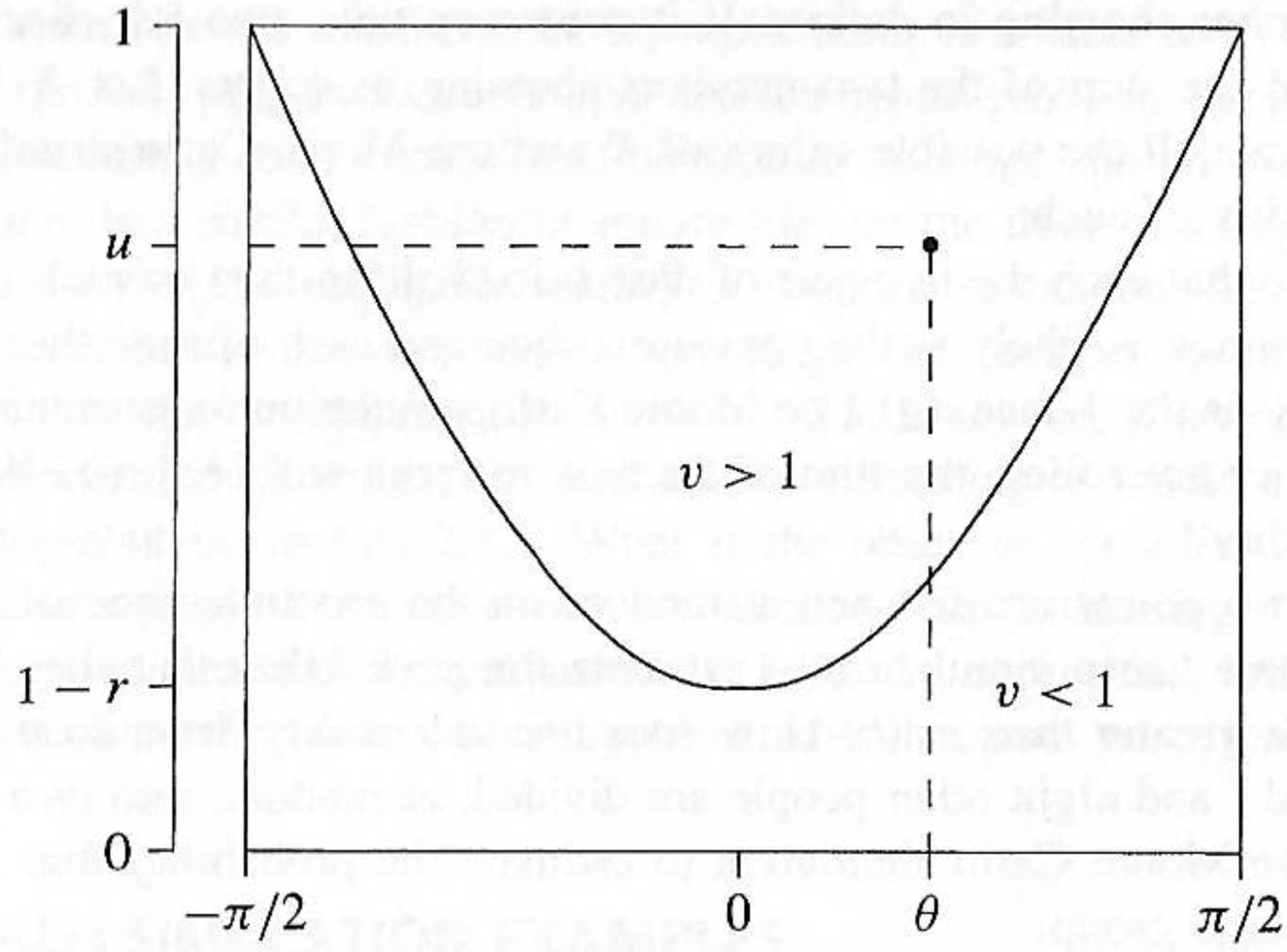

Figure 2.3.7 Sample space of all (θ, u) values.

The probability that the needle will cross at least one line is equivalent to the probability that a (θ, u) point (indicated by the '•') will fall in the shaded region with the curved boundary defined by the equation $u = 1 - r\cos(\theta)$. Because all points within the rectangle are equally likely, the probability of interest is the area of the shaded region divided by the area of the rectangle. If $0 < r \leq 1$, as illustrated, then the area of the shaded region is

$$\pi - \int_{-\pi/2}^{\pi/2} \left(1 - r\cos\theta\right) d\theta = r \int_{-\pi/2}^{\pi/2} \cos\theta \, d\theta = \cdots = 2r.$$

Therefore, because the area of the rectangle is π, the probability that the needle will cross at least one line is $2r/\pi$. The case $r > 1$ is left as an exercise.

Disclaimer. The simplicity of programs `galileo` and `buffon` is compelling. This simplicity is possible because the associated static models are so simple. That simplicity makes it possible to effectively short-circuit the first three model-development steps in Algorithm 1.1.1. Do not be misled by this into thinking that a similar short-circuiting is desirable for more complicated static systems or any *dynamic* stochastic system.

2.3.6 Exercises

2.3.1 (*a*) Derive the theoretical value of the probability of Buffon's needle crossing at least one line when $r > 1$. (*b*) Use Monte Carlo simulation to verify the correctness of your result when $r = 2$.

2.3.2 A rod of length 1 is broken at random into three pieces. (*a*) Use Monte Carlo simulation to estimate the probability that the resulting three pieces form a triangle. (*b*) Explain your interpretation of "at random" in this case. (*c*) What is the axiomatic probability?

2.3.3 A fair coin is tossed once. If it comes up heads, a fair die is rolled, and you are paid the number showing in dollars. If it comes up tails, two fair dice are rolled, and you are paid the sum of the two numbers showing in dollars. Let X be the amount won. Enumerate all the possible values of X and use Monte Carlo simulation to estimate the probability of each.

2.3.4 Suppose that each die in a pair of dice is loaded (un-fair) in such a way that the 6-face is four times as likely as the opposite 1-face and each of the other four faces are twice as likely as the 1-face. (*a*) Use Monte Carlo simulation to estimate the probability that, if the dice are rolled, the sum of the two up-faces will be 7. (*b*) What is the axiomatic probability?

2.3.5 (*a*) If two points are selected at random on the circumference of a circle of radius ρ, use Monte Carlo simulation to estimate the probability that the distance between the points is greater than ρ. (*b*) How does this probability depend on ρ?

2.3.6 You and I and eight other people are divided, at random, into two groups, each of size five. Use Monte Carlo simulation to estimate the probability that we will both end up in the same group.

2.3.7 Three dice are rolled and the largest of the three up faces is recorded. Let X be this value. The possible values of X are $1, 2, \ldots, 6$. Use Monte Carlo simulation to estimate the probability of each possible value.

2.3.8 Three identical boxes each contain two compartments. The first box has a \$10 bill in each compartment. The second box has a \$5 bill in each compartment. The third box has a \$10 bill in one compartment and a \$5 bill in the other compartment. A box is selected at random and one of compartments of that box is selected at random and is opened. If the compartment contains a \$10 bill, use Monte Carlo simulation to estimate the probability that the other compartment also contains a \$10 bill.

2.3.9 (*a*) Correct the algorithm in Example 2.3.8. (*b*) How would you test this new algorithm for correctness? *Hint*: The assignment `r = Uniform(0, ρ)` must be altered to make large values of r more likely than small values.

2.3.10 (*a*) Use Monte Carlo simulation to estimate the probability in Example 2.3.2. (*b*) Verify that the axiomatic approach yields the same probability. (*c*) How does this probability depend on a and b?

2.3.11 (*a*) Modify program `galileo` to estimate the probability of rolling a sum of 8 when five fair dice are tossed. (*b*) Compare the estimated probability with the value obtained by the axiomatic method.

2.3.12 Consider the intuitive (and incorrect) algorithm for generating a point uniformly in the interior of a circle given in Example 2.3.8. Using this algorithm, find the probability of being within a of the center of the circle, where $0 < a < \rho$. If the point generated was truly a random point in the interior of the circle, what should this probability equal?

2.3.13 Use program `buffon` to estimate π.

2.3.14 Use Monte Carlo simulation to estimate the average distance between two points generated randomly in the interior of a unit square.

2.3.15 Use Monte Carlo simulation to estimate the average shortest circuit (a path that begins and ends at one arbitrary point) among four points generated randomly in the interior of a unit square.

2.3.16 Modify program `buffon` to estimate π 30 times, using 10 000 needle tosses per replication. Summarize the results of the 30 replications in a table or an appropriate figure.

2.3.17 Consider the following *two-dimensional* Buffon needle problem. Suppose that an infinite family of infinitely long vertical and horizontal lines are spaced one unit apart in the (x, y) plane, in a similar fashion to square tiles on the floor of a large room. A needle of length $r < 1$ (and negligible width) is dropped at random onto the plane. If an observer considered only the vertical crossings, the method for estimating π would be identical to the development in Section 2.3.5. Likewise, if an observer considered only the horizontal crossings, the method for estimating π would also be identical to the development in Section 2.3.5. What if the observer considered both vertical and horizontal crossings? Develop a conceptual model for estimating π when both types of crossings are considered simultaneously. Is this an appropriate technique for halving the number of needle tosses necessary to achieve the same precision?

2.4 MONTE CARLO SIMULATION EXAMPLES

This section presents four applications of Monte Carlo simulation that are designed to augment the elementary examples considered in Section 2.3. The applications have been chosen to highlight the diversity of problems that can be addressed by Monte Carlo simulation and have been arranged in increasing order of the complexity of their implementation. The problems are (1) estimating the probability that the determinant of a 3×3 matrix of random numbers having a particular sign pattern is positive (program `det`), (2) estimating the probability of winning in Craps (program `craps`), (3) estimating the probability that a hatcheck girl will return all of the hats to the wrong owners when she returns n hats at random (program `hat`) and (4) estimating the mean time to complete a stochastic activity network (program `san`). Although the axiomatic approach to probability can be used to solve some of these problems, a minor twist in the assumptions associated with the problems often sinks an elegant axiomatic solution. A minor twist in the assumptions typically does not cause serious difficulties with the Monte Carlo simulation approach.

2.4.1 Random Matrices

Although the elementary definitions associated with matrices may be familiar to readers who have taken a course in linear algebra, we begin by reviewing some elementary definitions associated with matrices.

Matrices and Determinants

Definition 2.4.1 A *matrix* is a collection, including possible repetitions, of real or complex numbers arranged in a rectangular array.