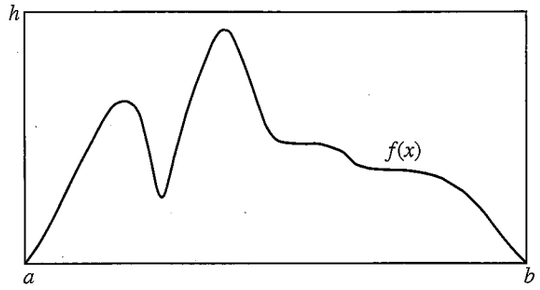

Figure 11.3 The function $f(x)$ is in the domain determined by the rectangle of height h and width $(b-a)$.

of $f(x)$ to the area of the rectangle. Hence, the estimate F_n in the *hit or miss* method is given by

$$F_n = A\frac{n_s}{n} \qquad \text{(hit or miss method)}, \tag{11.13}$$

where n_s is the number of points below the curve or "splashes," and n is the total number of points. The number of points chosen at random in (11.13) should not be confused with the number of intervals used in the numerical methods discussed in Section 11.1.

Another Monte Carlo integration method is based on a mean-value theorem of integral calculus which states that the definite integral (11.1) is determined by the average value of the integrand $f(x)$ in the range $a \le x \le b$:

$$F = \int_a^b f(x)\,dx = (b-a)\langle f\rangle. \tag{11.14}$$

To determine $\langle f\rangle$, we choose the x_i at random instead of at regular intervals and *sample* the values of $f(x)$. For the one-dimensional integral (11.1), the estimate F_n of the integral in the *sample mean* method is given by

$$F_n = (b-a)\frac{1}{n}\sum_{i=1}^{n} f(x_i) \approx (b-a)\langle f\rangle \quad \text{(sample mean method)}. \tag{11.15}$$

The x_i are random numbers distributed uniformly in the interval $a \le x_i \le b$, and n is the number of samples. Note that the forms of (11.3) and (11.15) are formally identical except that the n points are chosen with equal spacing in (11.3) and with random spacing in (11.15). We will find that for low-dimensional integrals, (11.3) is more accurate, but for higher-dimensional integrals, (11.15) does better.

A simple program that implements the hit or miss method is given in Listing 11.4. Note the use of the `Random` class and the methods `setSeed` and `nextDouble()`. As discussed in Chapter 7, the primary reason that it is desirable to specify the seed rather than to choose it more or less at random from the time (as is done by `Math.random()`) is that it is convenient to use the same random number sequence when testing a Monte Carlo program. Suppose that your program gives a strange result for a particular run. If we find an error in the program, we can use the same random number sequence to test whether our program changes make

any difference. Another reason for specifying the seed is so that other users can reproduce your results.

Listing 11.4 `MonteCarloEstimatorApp` displays the estimate of the integral for the number of samples equal to 2^p.

```
package org.opensourcephysics.sip.ch11;
import java.util.Random;
import org.opensourcephysics.controls.*;

public class MonteCarloEstimatorApp extends AbstractSimulation {
   Random rnd = new Random();
   int nSampled; // number of points already sampled
   int nTotal;   // total number of samples
   long seed;
   double a, b;  // interval limits
   double ymax;
   int hits = 0;

   public void reset() {
      control.setValue("lower limit a", 0);
      control.setValue("upper limit b", 1.0);
      control.setValue("upper limit on y", 1.0);
      control.setValue("seed", 137933);
   }

   public double evaluate(double x) {
      return Math.sqrt(1-x*x);
   }

   public void initialize() {
      a = control.getDouble("lower limit a");
      b = control.getDouble("upper limit b");
      ymax = control.getDouble("upper limit on y");
      nSampled = 0;
      nTotal = 2;
      seed = (long) control.getInt("seed");
      hits = 0;
      rnd.setSeed(seed);
   }

   public void doStep() {
      // nextDouble returns random double between 0 (inclusive) and 1
      // (exclusive)
      for(int i = nSampled;i<nTotal;i++) {
         double x = a+rnd.nextDouble()*(b-a);
         double y = rnd.nextDouble()*ymax;
         if(y<=evaluate(x)) {
            hits++;
         }
      }
      control.println("# of samples = "+nTotal+" estimated area = "
                      +(hits*(b-a)*ymax)/nTotal);
      nSampled = nTotal; // number of points sample so far
      // increase # of samples by factor of 2 for next call to do step
      nTotal *= 2;
   }
```