Chapter 11

Numerical Integration and Monte Carlo Methods

10 April 2001

Simple classical and Monte Carlo methods are illustrated in the context of the numerical evaluation of definite integrals.

11.1 Numerical Integration Methods in One Dimension

Monte Carlo methods were introduced in Chapter 7 in the context of systems that are intrinsically random. In this chapter we will find that we can use sequences of random numbers to estimate definite integrals, a problem that seemingly has nothing to do with randomness. To place the Monte Carlo numerical integration methods in perspective, we will first discuss several common classical methods of determining the numerical value of definite integrals. We will see that these classical methods, although usually preferable in low dimensions, are impractical for multidimensional integrals and that Monte Carlo methods are essential for the evaluation of the latter if the number of dimensions is sufficiently high.

Consider the one-dimensional definite integral of the form

$$F = \int_a^b f(x)\,dx. \tag{11.1}$$

For some choices of the integrand $f(x)$, the integration in (11.1) can be done analytically, found in tables of integrals, or evaluated as a series. However, there are relatively few functions that can be evaluated analytically and most functions and must be integrated numerically.

The classical methods of numerical integration are based on the geometrical interpretation of the integral (11.1) as the area under the curve of the function $f(x)$ from $x = a$ to $x = b$ (see

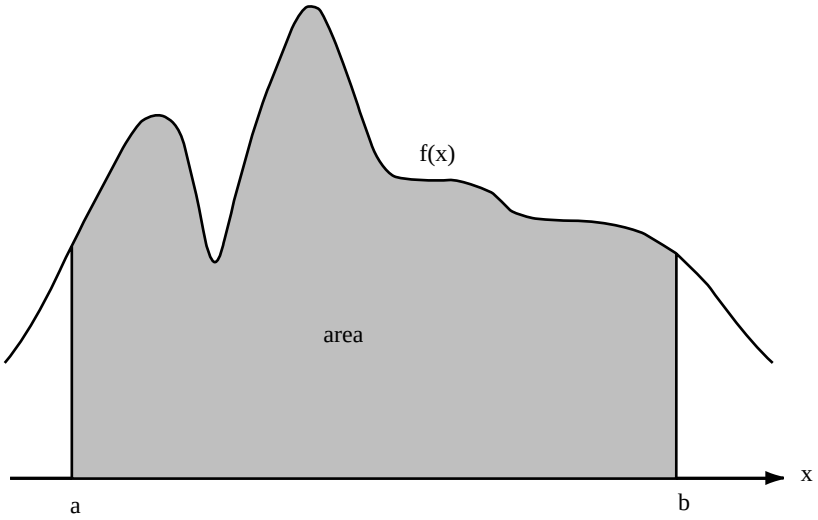

Figure 11.1: The integral F equals the area under the curve $f(x)$.

Figure 11.1). In these methods the x-axis is divided into n equal intervals of width Δx, where Δx is given by

$$\Delta x = \frac{b-a}{n}, \tag{11.2a}$$

and

$$x_n = x_0 + n\,\Delta x. \tag{11.2b}$$

In the above, $x_0 = a$ and $x_n = b$.

The simplest approximation of the area under the curve $f(x)$ is the sum of rectangles shown in Figure 11.2. In the *rectangular* approximation, $f(x)$ is evaluated at the *beginning* of the interval, and the approximate F_n of the integral is given by

$$F_n = \sum_{i=0}^{n-1} f(x_i)\Delta x. \qquad \text{(rectangular approximation)} \tag{11.3}$$

In the *trapezoidal* approximation or rule the integral is approximated by computing the area under a trapezoid with one side equal to $f(x)$ at the beginning of the interval and the other side equal to $f(x)$ at the end of the interval. This approximation is equivalent to replacing the function by a straight line connecting the values of $f(x)$ at the beginning and the end of each interval. Because the approximate area under the curve from x_i to x_{i+1} is given by $\frac{1}{2}[f(x_{i+1}) + f(x_i)]\Delta x$, the total area F_n is given by

$$F_n = \left[\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n)\right]\Delta x. \quad \text{(trapezoidal rule)} \tag{11.4}$$

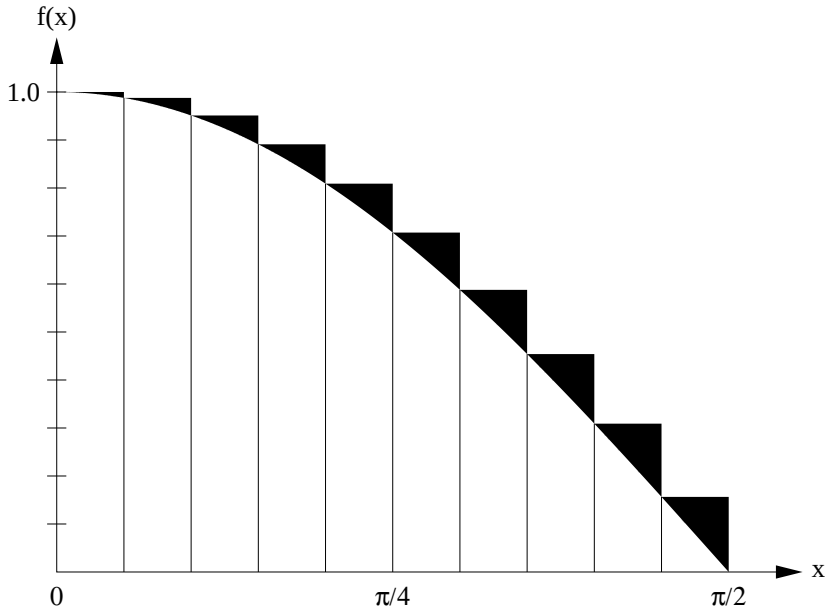

Figure 11.2: The rectangular approximation for $f(x) = \cos x$ for $0 \le x \le \pi/2$. The error in the rectangular approximation is shaded. Numerical values of the error for various values of the number of intervals n are given in Table 11.1.

A generally more accurate method is to use a quadratic or parabolic interpolation procedure through adjacent triplets of points. For example, the equation of the second-order polynomial that passes through the points (x_0, y_0), (x_1, y_1), and (x_2, y_2) can be written as

$$\begin{aligned} y(x) \;=\; & y_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} \\ & + y_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}. \end{aligned} \tag{11.5}$$

What is the value of $y(x)$ at $x = x_1$? The area under the parabola $y(x)$ between x_0 and x_2 can be found by simple integration and is given by

$$F_0 = \frac{1}{3}\,(y_0 + 4y_1 + y_2)\;\Delta x, \tag{11.6}$$

where $\Delta x = x_1 - x_0 = x_2 - x_1$. The total area under all the parabolic segments yields the parabolic approximation for the total area:

$$\begin{aligned} F_n \;=\; & \frac{1}{3}\big[f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + \ldots \\ & + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\big]\,\Delta x. \qquad \text{(Simpson's rule)} \end{aligned} \tag{11.7}$$

This approximation is known as *Simpson's rule*. Note that Simpson's rule requires that n be even.

In practice, Simpson's rule is adequate for functions $f(x)$ that are reasonably well behaved, that is, functions that can be adequately represented by a polynomial. If $f(x)$ is such a function, we can adopt the strategy of evaluating the area for a given number of intervals n and then doubling the number of intervals and evaluating the area again. If the two evaluations are sufficiently close to one another, we may stop. Otherwise, we again double n until we achieve the desired accuracy. Of course, this strategy will fail if $f(x)$ is not well-behaved. An example of a poorly-behaved function is $f(x) = x^{-1/3}$ at $x = 0$, where $f(x)$ diverges. Another example where this strategy might fail is when a limit of integration is equal to $\pm\infty$. In many cases we can eliminate the problem by a change of variables.

We first list classes `NumericalIntegration` and `RectangularApproximator`. What are the main features of `NumericalIntegration`?

```
// numerical integration of f(x) from x = a to x = b
package edu.clarku.sip.chapter11;
import edu.clarku.sip.templates.*;
import edu.clarku.sip.graphics.*;
import java.awt.*;
import java.awt.geom.*;
import java.text.NumberFormat;

public class NumericalIntegration implements Model
{
   private Function cosineFunction;
   private FunctionDrawer functionDrawer;        // draw function
   // compute sum and draw rectangles
   private RectangularApproximator rectangularApproximator;
   private Control myControl = new SControl(this);
   private World2D myWorld = new World2D();
   private NumberFormat numberFormat = NumberFormat.getInstance();

   public NumericalIntegration()
   {
      cosineFunction = new CosineFunction();
      rectangularApproximator = new RectangularApproximator(cosineFunction);
      functionDrawer = new FunctionDrawer(cosineFunction);
      myWorld.addDrawable(functionDrawer);
      myWorld.addDrawable(rectangularApproximator);
      numberFormat.setMaximumFractionDigits(4);
   }

   public void reset()
   {
      myControl.setValue("lower limit a", 0);
      myControl.setValue("upper limit b",  numberFormat.format(Math.PI/2));
```

```
      myControl.setValue("n", 4);
   }

   public void calculate()
   {
      double a = myControl.getValue("lower limit a");
      double b = myControl.getValue("upper limit b");
      int n = (int) myControl.getValue("n");     // number of intervals
      double dx = (b - a)/n; // width of interval
      functionDrawer.initialize(a, b, 0.01, false);
      double ymin = 0;
      double ymax = 1;
      myWorld.setXYMinMax(a, b, ymin, ymax);
      double areaUnderCurve = rectangularApproximator.computeArea(a, b, dx);
      myWorld.repaint();
      myControl.println("area under curve = " + areaUnderCurve);
   }

   public static void main(String[] args)
   {
      NumericalIntegration integ = new NumericalIntegration();
      integ.reset();
   }
}
```

The new classes that we have used in class `RectangularApproximator` are `Vector` and `Rectangle2D`. The `Vector` class is part of the java.util package which provides various container classes for storing and managing objects. Vectors can be thought of as dynamic arrays that can change size while the program is running. They are used in the following to store the rectangles that will be drawn. Vectors are less efficient than arrays and the elements in a vector must be objects. The simplest way to store an object in a `Vector` is to use the `add()` method.

```
package edu.clarku.sip.chapter11;
import java.awt.*;
import java.awt.geom.*;
import java.util.Vector;
import edu.clarku.sip.graphics.*;

public class RectangularApproximator implements Drawable
{
   private Function function;
   private Vector rectangles = new Vector();

   public RectangularApproximator(Function f)
   {
    function = f;
```

```
   }

   public double computeArea(double a, double b, double dx)
   {
      rectangles.clear();
      double x = a;
      double sum = function.evaluate(x);
      while (x < b)
      {
         double ytop = function.evaluate(x);
         double xleft = x;
         double height = ytop;
         double width = dx;
         // Rectangle is (already defined) Shape in Java 2D
         /* coordinates of rectangle are world coordinates; must convert
            them to pixel coordinates before drawing */
         Rectangle2D.Double r = new Rectangle2D.Double(xleft, ytop, width, height);
         rectangles.add(r);     // add rectangle to vector of rectangles
         x = x + dx;
         sum = sum + function.evaluate(x);
      }
      return sum*dx;
   }

   public void draw(World2D myWorld, Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;
      g2.setColor(Color.red);
      Rectangle2D.Double worldRect = null; // Rectangle in world coordinates
      Rectangle2D.Double pixRect = null;   // Rectangle in pixel coordinates
      // draw rectangles
      for (int i = 0; i < rectangles.size(); i++)
      {
         worldRect = (Rectangle2D.Double) rectangles.get(i); // get Rectangle from Vector
         double px = myWorld.xToPix(worldRect.x);
         double py = myWorld.yToPix(worldRect.y);
         double pwidth = myWorld.xToPix(worldRect.x + worldRect.width) - px;
         double pheight = myWorld.yToPix(0) - myWorld.yToPix(worldRect.height);
         pixRect = new Rectangle2D.Double( px, py, pwidth, pheight);
         g2.draw(pixRect);
      }
   }
}
```

The class `FunctionDrawer` defines a object that can draw itself using an object of type `GeneralPath`. The latter represents a geometric path constructed from straight lines, quadratic, and cubic curves.

What is the function of each method that was used? An affine transform is used to transform the coordinate system.

```
package edu.clarku.sip.chapter11;
import edu.clarku.sip.graphics.*;
import java.awt.*;
import java.awt.geom.*;
// class draws a function from xmin to xmax

public class FunctionDrawer implements Drawable
{
   private double ymin, ymax; // y min and max values of function in specified range
   // GeneralPath class represents a geometric path constructed from straight
   // lines, quadratic, and cubic curves
   private GeneralPath path = new GeneralPath();
   private Function function;
   private boolean filled = false;   // fill area under curve with drawing color
   public Color color = Color.black;

   public FunctionDrawer(Function f)
   {
      function = f;
   }

   public double getYMin(){return ymin;}

   public double getYMax(){return ymax;}

   public void initialize(double xmin, double xmax, double stepSize, boolean _filled)
   {
      filled = _filled;
      // reset path to empty
      path.reset();
      path.moveTo((float) xmin, (float) function.evaluate(xmin));
      ymin = function.evaluate(xmin);
      ymax = ymin; // starting values for ymin and ymax
      if (filled)
      {
         path.moveTo((float) xmin, 0);
         path.lineTo((float) xmin, (float) ymin);
      }
      else
         path.moveTo((float)xmin, (float)ymin);
      for (double x = xmin + stepSize; x <= xmax; x = x + stepSize)
      {
         // add point to path by drawing straight line from current to specified coordinates
```

```
         double y = function.evaluate(x);
         path.lineTo((float) x,(float) y);
         ymin = Math.min(ymin, y);
         ymax = Math.max(ymax, y);
      }
      if (filled)
      {
         path.lineTo((float)xmax, 0);
         path.closePath();
       }
   }

   public void draw(World2D myWorld, Graphics g)
   {
      Graphics2D g2 = (Graphics2D) g;
      g2.setColor(color);
      /* AffineTransform class performs linear mapping from 2D coordinates
         to other 2D coordinates preserving straightness and parallelness of lines */
      /* AffineTransform(double m00,double m10,double m01,double m11,double m02,double m12)
        [x']   [m00  m01  m02] [x]   [m00x + m01y + m02]
        [y'] = [m10  m11  m12] [y] = [m10x + m11y + m12]
        [1 ]   [ 0    0    1 ] [1]   [         1       ] */
      double m00 = myWorld.getXPixPerUnit();
      double m11 = -myWorld.getYPixPerUnit();
      double m02 = myWorld.getXOffset();
      double m12 = myWorld.getSize().height - myWorld.getYOffset();
      AffineTransform transform = new AffineTransform(m00, 0, 0, m11, m02, m12);
      /* return new Shape object defined by geometry of specified Shape after
         it has been transformed */
      Shape s = path.createTransformedShape(transform);
      if (filled)
         g2.fill(s);
      else
         g2.draw(s);
   }
}

package edu.clarku.sip.chapter11;
public class CosineFunction implements Function
{
   public double evaluate(double x)
   {
      return Math.cos(x);
   }
}
```

```
package edu.clarku.sip.chapter11;
public interface Function
{
   public double evaluate(double x);
}
```

Let us consider the accuracy of the rectangular approximation for the integral of $f(x) = \cos x$ from $x = 0$ to $x = \pi/2$ by comparing the numerical results shown in Table 11.1 with the exact answer of unity. Note that the error decreases as n^{-1}. This observed n dependence of the error is consistent with the analytical derivation of the n dependence of the error obtained in Appendix 11A. We explore the n dependence of the error associated with other numerical integration methods in Problems 11.1 and 11.2.

n	F_n	Δ_n
2	1.34076	0.34076
4	1.18347	0.18347
8	1.09496	0.09496
16	1.04828	0.04828
32	1.02434	0.02434
64	1.01222	0.01222
128	1.00612	0.00612
256	1.00306	0.00306
512	1.00153	0.00153
1024	1.00077	0.00077

Table 11.1: Rectangular approximations of the integral of $\cos x$ from $x = 0$ to $x = \pi/2$ as a function of n, the number of intervals. The error Δ_n is the difference between the rectangular approximation and the exact result of unity. Note that the error Δ_n decreases approximately as n^{-1}, that is, if n is increased by a factor of 2, Δ_n decreases by a factor 2.

Problem 11.1. The rectangular and midpoint approximations

a. Test the above program by reproducing the results in Table 11.1.

b. Use the rectangular approximation to determine numerical approximations for the definite integrals of $f(x) = 2x + 3x^2 + 4x^3$ and $f(x) = e^{-x}$ for $0 \le x \le 1$ and $f(x) = 1/x$ for $a \le x \le 2$. What is the approximate n dependence of the error in each case?

c. A straightforward modification of the rectangular approximation is to evaluate $f(x)$ at the *midpoint* of each interval. Define a `MidpointApproximator` class by making the necessary modifications and approximate the integral of $f(x) = \cos x$ in the interval $0 \le x \le \pi/2$. How does the magnitude of the error compare with the results shown in Table 11.1? What is the approximate dependence of the error on n?

d. Use the midpoint approximation to determine the definite integrals considered in part (b). What is the approximate n dependence of the error in each case? Given that our goal is to compute

the integral as accurately as possible with the smallest number of function evaluations of the integrand, which approximation would you choose?

Problem 11.2. The trapezoidal and Simpson's rule

a. Modify your program so that the trapezoidal rule is computed and determine the n-dependence of the error for the same functions as in Problem (11.1b).

b. It is possible to double the number of intervals without losing the benefit of the previous calculations. For $n = 1$, the trapezoidal rule is proportional to the average of $f(a)$ and $f(b)$. In the next approximation the value of f at the midpoint is added to this average. The next refinement adds the values of f at the $1/4$ and $3/4$ points. Modify your program so that the number of intervals is doubled each time and the results of previous calculations are used. The following pseudocode should be helpful:

```
if (n == 1) then
   sum = 0.5*(b - a)*(f(a) + f(b));
else
{
   int nadd = (int) Math.pow(2.0, n);   // additional intervals
   double delta = (b - a)/nadd;
   double x = a + 0.5*delta;
   double sumit = 0.0;       // intermediate sum
   for (int i = 1; i <= nadd; i++)
   {
      sumit = sumit + f(x);
      x = x + delta;
   }
   sum = 0.5*(sum + (b - a)*sum/nadd;
}
```

c. Use the trapezoidal approximation to approximate the integrals of $f(x) = \cos x$ for $0 \leq x \leq \pi/2$ and $f(x) = 2x + 3x^2 + 4x^3$ and $f(x) = e^{-x}$ for $0 \leq x \leq 1$. What is the approximate n dependence of the error in each case? Which approximation yields the best results for the same computation time?

d. Either adapt your program so that it uses Simpson's rule directly or convince yourself that the result of Simpson's rule can be expressed as

$$S_n = (4T_{2n} - T_n)/3, \tag{11.8}$$

where T_n is the result from the trapezoidal rule for n intervals. Determine the same integrals as in part (c) and discuss the relative merits of the various approximations.

e. Use Simpson's rule to approximate the integral of $f(x) = (2\pi)^{-1/2}\, e^{-x^2}$ for $-1 \leq x \leq 1$. Do you obtain the same result by choosing the interval $[0, 1]$ and then multiplying by two? Why or why not?

f. Evaluate the integral of the function $f(x) = 4\sqrt{1-x^2}$ for $-1 \leq x \leq 1$. What value of n is needed for four decimal accuracy? The reason for the slow convergence can be understood by reading Appendix 11A.

g. So far, our strategy for numerically estimating the value of definite integrals has been to choose one or more of the classical integration formulae and to compute F_n and F_{2n} for reasonable values of n. If the difference $|F_{2n} - F_n|$ is too large, then we double n until the desired accuracy is reached. The success of this strategy is based on the implicit assumption that the sequence $F_n, F_{2n}, \cdots$ converges to the true integral F. Is there a way of extrapolating this sequence to the limit? Let us explore this idea by using the trapezoidal approximation. Because the error for this approximation decreases approximately as n^{-2}, we can write $F = F_n + Cn^{-2}$, and plot F_n as a function of n^{-2} to obtain the extrapolated result F. Apply this procedure to the integrals considered in some of the above problems and compare your results to those found from the trapezoidal approximation and Simpson's rule alone. A more sophisticated application of this idea is known as *Romberg* integration (cf. Press et al.).

11.2 Simple Monte Carlo Evaluation of Integrals

We now explore a totally different method of estimating integrals. Let us introduce this method by asking, ' Suppose the pond is in the middle of a field of known area A. One way to estimate the area of the pond is to throw the stones so that they land at random within the boundary of the field and count the number of splashes that occur when a stone lands in a pond. The area of the pond is approximately the area of the field times the fraction of stones that make a splash. This simple procedure is an example of a *Monte Carlo* method.

More explicitly, imagine a rectangle of height h, width $(b-a)$, and area $A = h(b-a)$ such that the function $f(x)$ is within the boundaries of the rectangle (see Figure 11.3). Compute n pairs of random numbers x_i and y_i with $a \leq x_i \leq b$ and $0 \leq y_i \leq h$. The fraction of points x_i, y_i that satisfy the condition $y_i \leq f(x_i)$ is an estimate of the ratio of the integral of $f(x)$ to the area of the rectangle. Hence, the estimate F_n in the *hit or miss* method is given by

$$F_n = A\frac{n_s}{n}, \qquad \text{(hit or miss method)} \tag{11.9}$$

where n_s is the number of "splashes" or points below the curve, and n is the total number of points. The number of *trials* n in (11.9) 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 the mean-value theorem of calculus, which states that the definite integral (11.1) is determined by the average value of the integrand $f(x)$ in the range $a \leq x \leq b$. To determine this average, we choose the x_i at random instead of at regular intervals and *sample* the value 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

$$F_n = (b-a)\langle f \rangle = (b-a)\frac{1}{n}\sum_{i=1}^{n} f(x_i). \qquad \text{(sample mean method)} \tag{11.10}$$

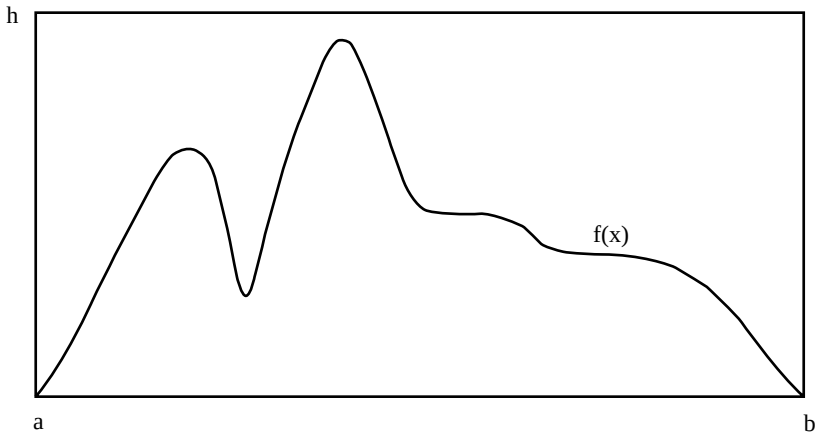

Figure 11.3: The function $f(x)$ is in the domain determined by the rectangle of height H and width $(b-a)$.

The x_i are random numbers distributed uniformly in the interval $a \leq x_i \leq b$, and n is the number of trials. Note that the forms of (11.3) and (11.10) are formally identical except that the n points are chosen with equal spacing in (11.3) and with random spacing in (11.10). We will find that for low dimensional integrals (11.3) is more accurate, but for higher dimensional integrals (11.10) does better.

A simple program that implements the hit or miss method is given below. Note the use of the `Random` class and the methods `setSeed` and `nextDouble()`. 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 done by `Math.random()`) is that it is convenient to use the same random number sequence when testing a Monte Carlo program. In Section **??**.

```
package edu.clarku.sip.chapter11;
import edu.clarku.sip.templates.*;
import java.util.Random;

public class MonteCarloEstimation implements Model
{
   private Control myControl = new SControl(this);
   private Random rnd = new Random();
   private int n;              // number of trials
   private long seed;

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

```
   public void calculate()
   {
      double ymax = 1.0;
      double a = 0;
      double b = 1.0;
      n = (int) myControl.getValue("n");
      seed = (long) myControl.getValue("seed");
      rnd.setSeed(seed);
      long hits = 0;
      double x, y;

      for (int i = 0; i < n; i++)
      {
         // nextDouble returns random double between 0 (inclusive) and 1 (exclusive)
         x = rnd.nextDouble()*(b-a);
         y = rnd.nextDouble()*ymax;
         if ( y <= evaluate(x) )
            hits++;
      }

      double estimatedArea = (hits*(b-a)*ymax)/n;
      myControl.println("estimated area = " + estimatedArea);
   }

   public void reset()
   {
      myControl.setValue("n", 1000);
      myControl.setValue("seed", 1379);
   }

   public static void main(String[] args)
   {
      new MonteCarloEstimation().reset();
   }
}
```

Problem 11.3. Monte Carlo integration in one dimension

a. Use the hit or miss Monte Carlo method to estimate F_n, the integral of $f(x) = 4\sqrt{1-x^2}$ in the interval $0 \leq x \leq 1$ as a function of n. Choose $a = 0$, $b = 1$, $h = 1$, and compute the mean value of the function $\sqrt{1-x^2}$. Multiply the estimate by 4 to determine F_n. Calculate the difference between F_n and the exact result of π. This difference is a measure of the error associated with the Monte Carlo estimate. Make a log-log plot of the error as a function of n. What is the approximate functional dependence of the error on n for large n, for example, $n \geq 10^4$?

b. Estimate the same integral using the sample mean Monte Carlo method (11.10) and compute

the error as a function of the number of trials n for $n \geq 10^4$. How many trials are needed to determine F_n to two decimal places? What is the approximate functional dependence of the error on n for large n?

c. Determine the computational time per trial using the two Monte Carlo methods. Which Monte Carlo method is preferable?

11.3 *Numerical Integration of Multidimensional Integrals

Many problems in physics involve averaging over many variables. For example, suppose we know the position and velocity dependence of the total energy of ten interacting particles. In three dimensions each particle has three velocity components and three position components. Hence the total energy is a function of 60 variables, and a calculation of the average energy per particle involves computing a $d = 60$ dimensional integral. (More accurately, the total energy is a function of $60 - 6 = 54$ variables if we use center of mass and relative coordinates.) If we divide each coordinate into p intervals, there would be p^{60} points to sum. Clearly, standard numerical methods such as Simpson's rule would be impractical for this example.

A discussion of the n dependence of the error associated with the standard numerical methods for d-dimensional integrals is given in Appendix 11A. We show that if the error decreases as n^{-a} for $d = 1$, then the error decreases as $n^{-a/d}$ in d dimensions. In contrast, we find (see Section 11.4) that the error for all Monte Carlo integration methods decreases as $n^{-1/2}$ independently of the dimension of the integral. Because the computational time is roughly proportional to n in both the classical and Monte Carlo methods, we conclude that for low dimensions, the classical numerical methods such as Simpson's rule are preferable to Monte Carlo methods unless the domain of integration is very complicated. However, the error in the conventional numerical methods increases with dimension (see Appendix 11A), and Monte Carlo methods are essential for higher dimensional integrals.

To illustrate the general method for evaluating multidimensional integrals, we consider the two-dimensional integral

$$F = \int_R f(x, y)\, dxdy, \tag{11.11}$$

where R denotes the region of integration. The extension to higher dimensions is straightforward, but tedious. Form a rectangle that encloses the region R, and divide this rectangle into squares of length h. Assume that the rectangle runs from x_a to x_b in the x direction and from y_a to y_b in the y direction. The total number of squares is $n_x n_y$, where $n_x = (x_b - x_a)/h$ and $n_y = (y_b - y_a)/h$. If we use the midpoint approximation, the integral F is estimated by

$$F \approx \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} f(x_i, y_j) H(x_i, y_j)\, h^2, \tag{11.12}$$

where $x_i = x_a + (i - \frac{1}{2})h$, $y_j = y_a + (j - \frac{1}{2})h$, and the function $H(x, y)$ equals unity if (x, y) is in R and is zero otherwise.

A simple Monte Carlo method for evaluating a two-dimensional integral uses the same rectangular region as in the above, but the n points (x_i, y_i) are chosen at random within the rectangle. The estimate for the integral is then

$$F_n = \frac{A}{n} \sum_{i=1}^{n} f(x_i, y_i) H(x_i, y_i), \tag{11.13}$$

where A is the area of the rectangle. Note that if $f(x, y) = 1$ everywhere, then (11.13) is equivalent to the hit or miss method of calculating the area of the region R. In general, (11.13) represents the area of the region R multiplied by the average value of $f(x, y)$ in R. In Section 11.7 we discuss a more efficient Monte Carlo method for evaluating definite integrals.

Problem 11.4. Two-dimensional numerical integration

a. Write a program to implement the midpoint approximation in two dimensions and integrate the function $f(x, y) = x^2 + 6xy + y^2$ over the region defined by the condition $x^2 + y^2 \leq 1$. Use $h = 0.1$, 0.05, 0.025, and if time permits 0.0125. Display n, the number of squares, and the estimate for the integral.

b. Repeat part (a) using a Monte Carlo method and the same number of points n. For each value of n repeat the calculation several times to obtain a crude estimate of the random error.

Problem 11.5. Volume of a hypersphere

a. The interior of a d-dimensional hypersphere of unit radius is defined by the condition ${x_1}^2 + {x_2}^2 + \ldots {x_d}^2 \leq 1$. Write a program that finds the volume of a hypersphere using the midpoint approximation. If you are clever, you can write a program that does any dimension using recursive subroutines. Test your program for $d = 2$ and $d = 3$, and then find the volume for $d = 4$ and $d = 5$. Begin with $h = 0.2$, and decrease h until your results do not change by more than 1%, or until you run out of patience or resources.

b. Repeat part (a) using a Monte Carlo technique. For each value of n, repeat the calculation several times to obtain a rough estimate of the random error. Is a program valid for any d easier to write in this case than in part (a)?

11.4 Monte Carlo Error Analysis

Both the classical numerical integration methods and the Monte Carlo methods yield approximate answers whose accuracy depends on the number of intervals or on the number of trials respectively. So far, we have used our knowledge of the exact value of various integrals to determine that the error in the Monte Carlo method approaches zero as approximately $n^{-1/2}$ for large n, where n is the number of trials. In the following, we will find how to estimate the error when the exact answer is unknown. Our main result is that the n dependence of the error is independent of the nature of the integrand and, most importantly, independent of the number of dimensions.

Because the appropriate measure of the error in Monte Carlo calculations is subtle, we first determine the error for an explicit example. Consider the Monte Carlo evaluation of the integral

of $f(x) = 4\sqrt{1-x^2}$ in the interval $[0,1]$ (see Problem 11.3). Our result for a particular sequence of $n = 10^4$ random numbers using the sample mean method is $F_n = 3.1489$. How does this result for F_n compare with your result found in Problem 11.3 for the same value of n? By comparing F_n to the exact result of $F = \pi \approx 3.1416$, we find that the error associated with $n = 10^4$ trials is approximately 0.0073.

How can we estimate the error if the exact result is unknown? How can we know if $n = 10^4$ trials is sufficient to achieve the desired accuracy? Of course, we cannot answer these questions definitively because if the actual error in F_n were known, we could correct F_n by the required amount and obtain F. The best we can do is to calculate the *probability* that the true value F is within a certain range centered on F_n.

If the integrand were a constant, then the error would be zero, that is, F_n would equal F for any n. Why? This limiting behavior suggests that a possible measure of the error is the *variance* σ^2 defined by

$$\sigma^2 = \langle f^2 \rangle - \langle f \rangle^2, \tag{11.14}$$

where

$$\langle f \rangle = \frac{1}{n} \sum_{i=1}^{n} f(x_i), \tag{11.15a}$$

and

$$\langle f^2 \rangle = \frac{1}{n} \sum_{i=1}^{n} f(x_i)^2. \tag{11.15b}$$

From the definition of the *standard deviation* σ, we see that if f is independent of x, σ is zero. For our example and the same sequence of random numbers used to obtain $F_n = 3.1489$, we obtain $\sigma_n = 0.8850$. Because this value of σ is two orders of magnitude larger than the actual error, we conclude that σ cannot be a direct measure of the error. Instead, σ is a measure of how much the function $f(x)$ varies in the interval of interest.

Another clue to finding an appropriate measure of the error can be found by increasing n and seeing how the actual error decreases as n increases. In Table 11.2 we see that as n goes from 10^2 to 10^4, the actual error decreases by a factor of 10, that is, as $\sim 1/n^{\frac{1}{2}}$. However, we also see that σ_n is roughly constant and is much larger than the actual error.

n	F_n	actual error	σ_n
10^2	3.0692	0.0724	0.8550
10^3	3.1704	0.0288	0.8790
10^4	3.1489	0.0073	0.8850

Table 11.2: Examples of Monte Carlo measurements of the mean value of $f(x) = 4\sqrt{1-x^2}$ in the interval $[0,1]$. The actual error is given by the difference $|F_n - \pi|$. The standard deviation σ_n is found using (11.14).

One way to obtain an estimate for the error is to make additional runs of n trials each. Each run of n trials yields a mean or *measurement* that we denote as M_α. In general, these

run α	M_α	actual error
1	3.1489	0.0073
2	3.1326	0.0090
3	3.1404	0.0012
4	3.1460	0.0044
5	3.1526	0.0110
6	3.1397	0.0019
7	3.1311	0.0105
8	3.1358	0.0058
9	3.1344	0.0072
10	3.1405	0.0011

Table 11.3: Examples of Monte Carlo measurements of the mean value of $f(x) = 4\sqrt{1-x^2}$ in the interval $[0, 1]$. A total of 10 measurements of $n = 10^4$ trials each were made. The mean value M_α and the actual error $|M_\alpha - \pi|$ for each measurement are shown.

measurements are not equal because each measurement uses a different finite sequence of random numbers. Table 11.3 shows the results of ten separate measurements of $n = 10^4$ trials each. We see that the actual error varies from measurement to measurement. Qualitatively, the magnitude of the differences between the measurements is similar to the actual errors, and hence these differences are a measure of the error associated with a single measurement. To obtain a quantitative measure of this error, we determine the differences of these measurements using the *standard deviation of the means* σ_m which is defined as

$$\sigma_m{}^2 = \langle M^2 \rangle - \langle M \rangle^2, \tag{11.16}$$

where

$$\langle M \rangle = \frac{1}{m} \sum_{\alpha=1}^{m} M_\alpha, \tag{11.17a}$$

and

$$\langle M^2 \rangle = \frac{1}{m} \sum_{\alpha=1}^{m} M_\alpha{}^2. \tag{11.17b}$$

From the values of M_α in Table 11.3 and the relation (11.16), we find that $\sigma_m = 0.0068$. This value of σ_m is consistent with the results for the actual errors shown in Table 11.3 which we see vary from 0.00112 to 0.01098. Hence we conclude that σ_m, the standard deviation of the means, is a measure of the error for a single measurement. The more precise interpretation of σ_m is that a single measurement has a 68% chance of being within σ_m of the "true" mean. Hence the probable error associated with our first measurement of F_n with $n = 10^4$ is 3.149 ± 0.007.

Although σ_m gives an estimate of the probable error, our method of obtaining σ_m by making additional measurements is impractical because we could have combined the additional measure-

ments to make a better estimate. In Appendix 11.8 we derive the relation

$$\sigma_m = \frac{\sigma}{\sqrt{n-1}} \tag{11.18a}$$

$$\approx \frac{\sigma}{\sqrt{n}}. \tag{11.18b}$$

The reason for the expression $1/\sqrt{n-1}$ in (11.18a) rather than $1/\sqrt{n}$ is similar to the reason for the expression $1/\sqrt{n-2}$ in the error estimates of the least squares fits (see (7.27c)). The idea is that to compute σ, we need to use n trials to compute the mean, $\langle f(x)\rangle$, and, loosely speaking, we have only $n-1$ independent trials remaining to calculate σ. Because we almost always make a large number of trials, we will use the relation (11.18b) and consider only this limit in Appendix 11A. Note that (11.18) implies that the most probable error decreases with the square root of the number of trials. For our example we find that the most probable error of our initial measurement is approximately $0.8850/100 \approx 0.009$, an estimate consistent with the known error of 0.007 and with our estimated value of $\sigma_m \approx 0.007$.

subset k	S_k
1	3.14326
2	3.15633
3	3.10940
4	3.15337
5	3.15352
6	3.11506
7	3.17989
8	3.12398
9	3.17565
10	3.17878

Table 11.4: The values of S_k for $f(x) = 4\sqrt{1-x^2}$ for $0 \le x \le 1$ is shown for 10 subsets of 10^3 trials each. The average value of $f(x)$ over the 10 subsets is 3.14892, in agreement with the result for F_n for the first measurement shown in Table 11.3.

One way to verify the relation (11.18) is to divide the initial measurement of n trials into s subsets. This procedure does not require additional measurements. We denote the mean value of $f(x_i)$ in the kth subset by S_k. As an example, we divide the 10^4 trials of the first measurement into $s = 10$ subsets of $n/s = 10^3$ trials each. The results for S_k are shown in Table 11.4. As expected, the mean values of $f(x)$ for each subset k are not equal. A reasonable candidate for a measure of the error is the standard deviation of the means of each subset. We denote this quantity as σ_s where

$$\sigma_s{}^2 = \langle S^2\rangle - \langle S\rangle^2, \tag{11.19}$$

where the averages are over the subsets. From Table 11.4 we obtain $\sigma_s = 0.025$, a result that is approximately three times larger than our estimate of 0.007 for σ_m. However, we would like to define an error estimate that is independent of how we subdivide the data. This quantity is not σ_s,

but the ratio $\sigma_s/\sqrt{s}$, which for our example is approximately $0.025/\sqrt{(10)} \approx 0.008$. This value is consistent with both σ_m and the ratio $\sigma/\sqrt{n}$. We conclude that we can interpret the n trials either as a single measurement or as a collection of s measurements with n/s trials each. In the former interpretation, the probable error is given by the standard deviation of the n trials divided by the square root of the number of trials. In the same spirit, the latter interpretation implies that the probable error is given by the standard deviation of the s measurements of the subsets divided by the square root of the number of measurements.

We can make the error as small as we wish by either increasing the number of trials or by increasing the efficiency of the individual trials and thereby reducing the standard deviation σ. Several *reduction of variance* methods are introduced in Sections 11.7 and 11.8.

Problem 11.6. Estimate of the Monte Carlo error

a. Estimate the integral of $f(x) = e^{-x}$ in the interval $0 \leq x \leq 1$ using the sample mean Monte Carlo method with $n = 10^2$, $n = 10^3$, and $n = 10^4$. Compute the standard deviation σ as defined by (11.14). Does your estimate of σ change significantly as n is increased? Determine the exact answer analytically and estimate the n dependence of the error. How does your estimated error compare with the error estimate obtained from the relation (11.18)?

b. Generate nineteen additional measurements of the integral each with $n = 10^3$ trials. Compute σ_m, the standard deviation of the twenty measurements. Is the magnitude of σ_m consistent with your estimate of the error obtained in part (a)? Will your estimate of σ_m change significantly if more measurements are made?

c. Divide your first measurement of $n = 10^3$ trials into $s = 20$ subsets of 50 trials each. Compute the standard deviation of the subsets σ_s. Is the magnitude $\sigma_s/\sqrt{s}$ consistent with your previous error estimates?

d. Divide your first measurement into $s = 10$ subsets of 100 trials each and again compute the standard deviation of the subsets. How does the value of σ_s compare to what you found in part (c)? What is the value of $\sigma_s/\sqrt{s}$ in this case? How does the standard deviation of the subsets compare using the two different divisions of the data?

e. Estimate the integral

$$\int_0^1 e^{-x^2}\, dx \tag{11.20}$$

to two decimal places using $\sigma_n/\sqrt{n}$ as an estimate of the probable error.

* *Problem 11.7.* Importance of randomness

We will learn in Chapter **??** that the random number generator included with many programming languages is based on the linear congruential method. In this method each term in the sequence can be found from the preceding one by the relation

$$x_{n+1} = (ax_n + c) \bmod m, \tag{11.21}$$

where x_0 is the seed, and a, c, and m are nonnegative integers. The random numbers r in the unit interval $0 \leq r < 1$ are given by $r_n = x_n/m$. The notation $y = x \bmod m$ means that if x exceeds m,

then the *modulus* m is subtracted from x as many times as necessary until $0 \leq y < m$. Eventually, the sequence of numbers generated by (11.21) will repeat itself, yielding a *period* for the random number generator. To examine the effect of a poor random number generator, we choose values of x_0, m, a, and c such that (11.21) has poor statistical properties, for example, a short period. What is the period for $x_0 = 1$, $a = 5$, $c = 0$, and $m = 32$? Estimate the integral in Problem a by making a single measurement of $n = 10^3$ trials using the "random number" generator (11.21) with the above values of x_0, a, c, and m. Analyze your measurement in the same way as before, that is, calculate the mean, the mean of each of the twenty subsets, and the standard deviation of the means of the subsets. Then divide your data into ten subsets and calculate the same quantities. Are the standard deviations of the subsets related as before? If not, why?

11.5 Nonuniform Probability Distributions

In the previous two sections we learned how uniformly distributed random numbers can be used to estimate definite integrals. We will find that it is desirable to sample the integrand $f(x)$ more often in regions of x where the magnitude of $f(x)$ is large or rapidly varying. Because such *importance sampling* methods require nonuniform probability distributions, we first consider several methods for generating random numbers that are not distributed uniformly before considering importance sampling in Section 11.7. In the following, we will denote r as a member of a uniform random number sequence in the unit interval $0 \leq r < 1$.

Suppose that two discrete events occur with probabilities p_1 and p_2 such that $p_1 + p_2 = 1$. How can we choose the two events with the correct probabilities using a uniform probability distribution? For this simple case, it is obvious that we choose event 1 if $r < p_1$; otherwise, we choose event 2. If there are three events with probabilities p_1, p_2, and p_3, then if $r < p_1$ we choose event 1; else if $r < p_1 + p_2$, we choose event 2; else we choose event 3. We can visualize these choices by dividing a line segment of unit length into three pieces whose lengths are as shown in Figure 11.4. A random point r on the line segment will land in the ith segment with a probability equal to p_i.

Now consider n discrete events. How do we determine which event, i, to choose given the value of r? The generalization of the procedure we have followed for $n = 2$ and 3 is to find the value of i that satisfies the condition

$$\sum_{j=0}^{i-1} p_j \leq r \leq \sum_{j=0}^{i} p_j, \tag{11.22}$$

where we have defined $p_0 \equiv 0$. Check that (11.22) reduces to the correct procedure for $n = 2$ and $n = 3$.

Now let us consider a continuous nonuniform probability distribution. One way to generate such a distribution is to take the limit of (11.22) and associate p_i with $p(x)\,dx$, the *probability density* $p(x)$, is defined such that $p(x)\,dx$ is the probability that the event x is in the interval between x and $x + dx$. The probability density $p(x)$ is normalized such that

$$\int_{-\infty}^{+\infty} p(x)\,dx = 1. \tag{11.23}$$

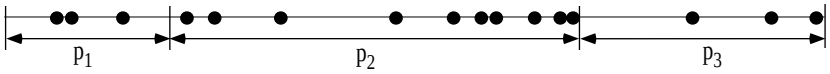

Figure 11.4: The unit interval is divided into three segments of lengths $p_1 = 0.2$, $p_2 = 0.5$, and $p_3 = 0.3$. Sixteen random numbers are represented by the filled circles uniformly distributed on the unit interval. The fraction of circles within each segment is approximately equal to the value of p_i for that segment.

In the continuum limit the two sums in (11.22) become the same integral and the inequalities become equalities. Hence we can write

$$P(x) \equiv \int_{-\infty}^{x} p(x')\,dx' = r. \tag{11.24}$$

From (11.24) we see that the uniform random number r corresponds to the *cumulative probability distribution* function $P(x)$, which is the probability of choosing a value less than or equal to x. The function $P(x)$ should not be confused with the probability density $p(x)$ or the probability $p(x)\,dx$. In many applications the meaningful range of values of x is positive. In that case, we have $p(x) = 0$ for $x < 0$.

The relation (11.24) leads to the *inverse transform* method for generating random numbers distributed according to the function $p(x)$. This method involves generating a random number r and solving (11.24) for the corresponding value of x. As an example of the method, we use (11.24) to generate a random number sequence according to the uniform probability distribution on the interval $a \le x \le b$. The desired probability density $p(x)$ is

$$p(x) = \begin{cases} (1/(b-a), & a \le x \le b \\ 0, & \text{otherwise.} \end{cases} \tag{11.25}$$

The cumulative probability distribution function $P(x)$ for $a \le x \le b$ can be found by substituting (11.25) into (11.24) and performing the integral. The result is

$$P(x) = \frac{x-a}{b-a}. \tag{11.26}$$

If we substitute the form (11.26) for $P(x)$ into (11.24) and solve for x, we find the desired relation

$$x = a + (b-a)r. \tag{11.27}$$

The variable x given by (11.27) is distributed according to the probability distribution $p(x)$ given by (11.25). Of course, the relation (11.27) is rather obvious, and we already have used (11.27) in our Monte Carlo programs.

We next apply the inverse transform method to the probability density function

$$p(x) = \begin{cases} (1/\lambda)\,e^{-x/\lambda}, & \text{if } 0 \le x \le \infty \\ 0, & x < 0\ . \end{cases} \tag{11.28}$$

In Section 11.6 we will use this probability density to find the distance between scattering events of a particle whose mean free path is λ. If we substitute (11.28) into (11.24) and integrate, we find the relation

$$r = P(x) = 1 - e^{-x/\lambda}. \tag{11.29}$$

The solution of (11.29) for x yields $x = -\lambda \ln(1-r)$. Because $1-r$ is distributed in the same way as r, we can write

$$x = -\lambda \ln r. \tag{11.30}$$

The variable x found from (11.30) is distributed according to the probability density $p(x)$ given by (11.28). On many computers the computation of the natural logarithm in (11.30) is relatively slow, and hence the inverse transform method might not necessarily be the most efficient method to use.

From the above examples, we see that two conditions must be satisfied in order to apply the inverse transform method. Specifically, the form of $p(x)$ must allow the integral in (11.24) to be performed analytically, and it must be feasible to invert the relation $P(x) = r$ for x. The Gaussian probability density

$$p(x) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-x^2/2\sigma^2} \tag{11.31}$$

is an example of a probability density for which the cumulative distribution $P(x)$ cannot be obtained analytically. However, we can generate the two-dimensional probability $p(x,y)\,dx\,dy$ given by

$$p(x,y)\,dx\,dy = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}\,dx\,dy. \tag{11.32}$$

First, we make a change of variables to polar coordinates:

$$r = (x^2+y^2)^{1/2} \qquad \theta = \tan^{-1}\frac{y}{x}. \tag{11.33}$$

Let $\rho = r^2/2$, and write the two-dimensional probability as

$$p(\rho,\theta)\,d\rho d\theta = \frac{1}{2\pi} e^{-\rho} d\rho\,d\theta, \tag{11.34}$$

where we have set $\sigma = 1$. If we generate ρ according to the exponential distribution (11.28) and generate θ uniformly in the interval $0 \le \theta < 2\pi$, then the variables

$$x = (2\rho)^{1/2}\cos\theta \quad \text{and} \quad y = (2\rho)^{1/2}\sin\theta \qquad \text{(Box-Muller method)} \tag{11.35}$$

will each be generated according to (11.31) with zero mean and $\sigma = 1$. (Note that the two-dimensional density (11.32) is the product of two independent one-dimensional Gaussian distributions.) This way of generating a Gaussian distribution is known as the *Box-Muller* method. We discuss other methods for generating the Gaussian distribution in Problem 11.12 and Appendix 11C.

Problem 11.8. Nonuniform probability densities

a. Write a program to simulate the simultaneous rolling of two dice. In this case the events are discrete and occur with nonuniform probability. You might wish to revisit Problem 7.12 and simulate the game of craps.

b. Write a program to verify that the sequence of random numbers $\{x_i\}$ generated by (11.30) is distributed according to the exponential distribution (11.28).

c. Generate random variables according to the probability density function

$$p(x) = \begin{cases} 2(1-x), & \text{if } 0 \le x \le 1; \\ 0, & \text{otherwise.} \end{cases} \tag{11.36}$$

d. Verify that the variables x and y in (11.35) are distributed according to the Gaussian distribution. What is the mean value and the standard deviation of x and of y?

e. How can you use the relations (11.35) to generate a Gaussian distribution with arbitrary mean and standard deviation?

11.6 *Neutron Transport

We now consider the application of a nonuniform probability distribution to the simulation of the transmission of neutrons through bulk matter, one of the original applications of a Monte Carlo method. Suppose that a neutron is incident on a plate of thickness t. We assume that the plate is infinite in the x and y directions and that the z axis is normal to the plate. At any point within the plate, the neutron can either be captured with probability p_c or scattered with probability p_s. These probabilities are proportional to the capture cross section and scattering cross section, respectively. If the neutron is scattered, we need to find its new direction as specified by the polar angle θ (see Figure 11.5). Because we are not interested in how far the neutron moves in the x or y direction, the value of the azimuthal angle ϕ is irrelevant.

If the neutrons are scattered equally in all directions, then the probability $p(\theta, \phi)\, d\theta d\phi$ equals $d\Omega/4\pi$, where $d\Omega$ is an infinitesimal solid angle and 4π is the total solid angle. Because $d\Omega = \sin\theta\, d\theta d\phi$, we have

$$p(\theta, \phi) = \frac{\sin\theta}{4\pi}. \tag{11.37}$$

We can find the probability density for θ and ϕ separately by integrating over the other angle. For example,

$$p(\theta) = \int_0^{2\pi} p(\theta, \phi)\, d\phi = \frac{1}{2}\sin\theta, \tag{11.38}$$

and

$$p(\phi) = \int_0^{\pi} p(\theta, \phi)\, d\theta = \frac{1}{2\pi}. \tag{11.39}$$

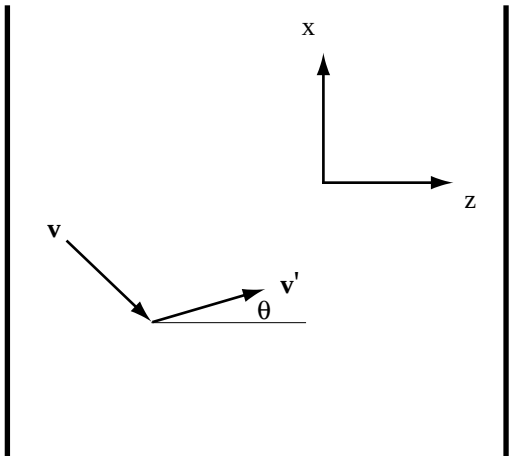

Figure 11.5: The definition of the scattering angle θ. The velocity before scattering is $\mathbf{v}$ and the velocity after scattering is $\mathbf{v}'$. The scattering angle θ is independent of $\mathbf{v}$ and is defined relative to the z axis.

Because the point probability $p(\theta, \phi)$ is the product of the probabilities $p(\theta)$ and $p(\phi)$, θ and ϕ are independent variables. Although we do not need to generate a random angle ϕ, we note that since $p(\phi)$ is a constant, ϕ can be found from the relation

$$\phi = 2\pi r. \tag{11.40}$$

To find θ according to the distribution (11.38), we substitute (11.38) in (11.24) and obtain

$$r = \frac{1}{2}\int_0^\theta \sin x\, dx \tag{11.41}$$

If we do the integration in (11.41), we find

$$\cos\theta = 1 - 2r. \tag{11.42}$$

Note that (11.40) implies that ϕ is uniformly distributed between 0 and 2π and (11.42) implies that $\cos\theta$ is uniformly distributed between -1 and $+1$.

We could invert the cosine in (11.42) to solve for θ. However, to find the z component of the path of the neutron through the plate, we need to multiply $\cos\theta$ by the path length ℓ, and hence we need $\cos\theta$ rather than θ. The path length, which is the distance traveled between subsequent scattering events, is obtained from the exponential probability density, $p(\ell) \propto e^{-\ell/\lambda}$ (see (11.28)). From (11.30), we have

$$\ell = -\lambda \ln r, \tag{11.43}$$

where λ is the mean free path.

Now we have all the necessary ingredients for calculating the probabilities for a neutron to pass through the plate, be reflected off the plate, or be captured and absorbed in the plate. The input parameters are the thickness of the plate t, the capture and scattering probabilities p_c and p_s, and the mean free path λ. We begin with $z = 0$, and implement the following steps:

1. Determine if the neutron is captured or scattered. If it is captured, then add one to the number of captured neutrons, and go to step 5.
2. If the neutron is scattered, compute $\cos\theta$ from (11.42) and ℓ from (11.43). Change the z coordinate of the neutron by $\ell\cos\theta$.
3. If $z < 0$, add one to the number of reflected neutrons. If $z > t$, add one to the number of transmitted neutrons. In either case, skip to step 5 below.
4. Repeat steps 1–3 until the fate of the neutron has been determined.
5. Repeat steps 1–4 with additional incident neutrons until sufficient data has been obtained.

Problem 11.9. Elastic neutron scattering

a. Write a program to implement the above algorithm for neutron scattering through a plate. Assume $t = 1$ and $p_c = p_s/2$. Find the transmission, reflection, and absorption probabilities for the mean free path λ equal to 0.01, 0.05, 0.1, 1, and 10. Begin with 100 incident neutrons, and increase this number until satisfactory statistics are obtained. Give a qualitative explanation of your results.

b. Choose $t = 1$, $p_c = p_s$, and $\lambda = 0.05$, and compare your results with the analogous case considered in part (a).

c. Repeat part (b) with $t = 2$ and $\lambda = 0.1$. Do the various probabilities depend on λ and t separately or only on their ratio? Answer this question before doing the simulation.

d. Draw some typical paths of the neutrons. From the nature of these paths, explain the results in parts (a)–(c). For example, how does the number of scattering events change as the absorption probability changes?

Problem 11.10. Inelastic neutron scattering

a. In Problem 11.9 we assumed elastic scattering, that is, no energy is lost during scattering. Here we assume that some of the neutron energy E is lost and that the mean free path is proportional to the speed and hence to $\sqrt{E}$. Modify your program so that a neutron loses a fraction f of its energy at each scattering event, and assume that $\lambda = \sqrt{E}$. Consider $f = 0.05, 0.1$, and 0.5, and compare your results with those found in Problem 11.9a.

b. Make a histogram for the path lengths between scattering events and plot the path length distribution function for $f = 0.1, 0.5$, and 0 (elastic scattering).

The above procedure for simulating neutron scattering and absorption is more computer intensive than necessary. Instead of considering a single neutron at a time, we can consider a collection of neutrons at each position. Then instead of determining whether one neutron is captured or scattered, we determine the fraction that is captured and the fraction that is scattered. For example, at the first scattering site, a fraction p_c of the neutrons are captured and a fraction p_s are scattered. We accumulate the fraction p_c for the captured neutrons. We also assume that all the scattered neutrons move in the same direction with the same path length, both of which are generated at random as before. At the next scattering site there are p_s^2 scattered neutrons and $p_s p_c$ captured neutrons. At the end of m steps, the fraction of neutrons remaining is $w = p_s^m$ and the total fraction of captured neutrons is $p_c + p_c p_s + p_c p_s^2 + \ldots + p_c p_s^{m-1}$. If the new position at the mth step is at $z < 0$, we add w to the sum for the reflected neutrons; if $z > t$, we add w to the neutrons transmitted. When the neutrons are reflected or absorbed, we start over again at $z = 0$ with another collection of neutrons.

Problem 11.11. Improved neutron scattering method Apply the improved Monte Carlo method to neutron transmission through a plate. Repeat the simulations suggested in Problem a and compare your new and previous results. Also compare the computational times for the two approaches to obtain comparable statistics.

The power of the Monte Carlo method becomes apparent when the geometry of the material is complicated or when the material is spatially nonuniform so that the cross sections vary from point to point. A difficult problem of current interest is the absorption of various forms of radiation in the human body.

Problem 11.12. Transmission through layered materials Consider two plates with the same thickness $t = 1$ that are stacked on top of one another with no space between them. For one plate, $p_c = p_s$, and for the other, $p_c = 2p_s$, that is, the top plate is a better absorber. Assume that $\lambda = 1$ in both plates. Find the transmission, reflection, and absorption probabilities for elastic scattering. Does it matter which plate receives the incident neutrons?

11.7 Importance Sampling

In Section 11.4 we found found that the error associated with a Monte Carlo estimate is proportional to σ and inversely proportional to the square root of the number of trials. Hence, there are only two ways of reducing the error in a Monte Carlo estimate – either increase the number of trials or reduce the variance. Clearly the latter choice is desirable because it not require much more computer time. In this section we introduce *importance sampling* techniques that reduce σ and improve the efficiency of each trial.

In the context of numerical integration, we introduce a positive function $p(x)$ such that

$$\int_a^b p(x)\,dx = 1, \tag{11.44}$$

and rewrite the integral (11.1) as

$$F = \int_a^b \left[\frac{f(x)}{p(x)}\right] p(x)\,dx. \tag{11.45}$$

We can evaluate the integral (11.45) by sampling according to the probability distribution $p(x)$ and constructing the sum

$$F_n = \frac{1}{n}\sum_{i=1}^{n}\frac{f(x_i)}{p(x_i)}. \tag{11.46}$$

The sum (11.46) reduces to (11.10) for the uniform case $p(x) = 1/(b-a)$.

We wish to choose a form for $p(x)$ that minimizes the variance of the integrand $f(x)/p(x)$. Because we cannot evaluate σ analytically in general, we determine σ *a posteriori* and choose a form of $p(x)$ that mimics $f(x)$ as much as possible, particularly where $f(x)$ is large. A suitable choice of $p(x)$ would make the integrand $f(x)/p(x)$ slowly varying, and hence the variance will be reduced. As an example, we again consider the integral (see Problem ee)

$$F = \int_0^1 e^{-x^2}\,dx. \tag{11.47}$$

The estimate of F with $p(x) = 1$ for $0 \le x \le 1$ is shown in the first column of Table 11.5. A reasonable choice of a weight function is $p(x) = Ae^{-x}$, where A is chosen such that $p(x)$ is normalized on the unit interval. Note that this choice of $p(x)$ is positive definite and is qualitatively similar to $f(x)$. The results are shown in the second column of Table 11.5. We see that although the computation time per trial for the nonuniform case is larger, the smaller value of σ makes the use of the nonuniform probability distribution more efficient.

	$p(x) = 1$	$p(x) = Ae^{-x}$
n (trials)	4×10^5	8×10^3
F_n	0.7471	0.7469
σ	0.2010	0.0550
$\sigma/\sqrt{n}$	3×10^{-4}	6×10^{-4}
Total CPU time (s)	35	1.35
CPU time per trial (s)	10^{-4}	2×10^{-4}

Table 11.5: Comparison of the Monte Carlo estimates of the integral (11.47) using the uniform probability density $p(x) = 1$ and the nonuniform probability density $p(x) = Ae^{-x}$. The normalization constant A is chosen such that $p(x)$ is normalized on the unit interval. The value of the integral to four decimal places is 0.7468. The estimates F_n, variance σ, and the probable error $\sigma/n^{1/2}$ are shown. The CPU time (seconds) is shown for comparison only.

Problem 11.13. Importance sampling

a. Choose $f(x) = \sqrt{1-x^2}$ as in the text and consider $p(x) = A(1-x)$ for $x \ge 0$. What is the value of A that normalizes $p(x)$ in the interval $[0, 1]$? What is the relation for the random variable x in terms of r assuming this form of the probability density $p(x)$? What is the variance of $f(x)/p(x)$ in the unit interval?

b. Choose the importance function $p(x) = Ae^{-x}$ and evaluate the integral

$$\int_0^3 x^{3/2}\,e^{-x}\,dx. \tag{11.48}$$

c. Choose $p(x) = Ae^{-\lambda x}$ and estimate the integral

$$\int_0^\pi \frac{1}{x^2 + \cos^2 x}\, dx. \tag{11.49}$$

Determine the value of λ that minimizes the variance of the integral.

An alternative approach is to use the known values of $f(x)$ at regular intervals to sample more often where $f(x)$ is relatively large. Because the idea is to use $f(x)$ itself to determine the probability of sampling, we only consider integrands $f(x)$ that are non-negative. To compute a rough estimate of the relative values of $f(x)$, we first compute its average value by taking k equally spaced points s_i and computing

$$S = \sum_{i=1}^{k} f(s_i) \tag{11.50}$$

This sum divided by k gives the average value of f in the interval. The approximate value of the integral is given by $F \approx Sh$, where $h = (b-a)/k$. This approximation of the integral is equivalent to the rectangular or mid-point approximation depending on where we compute the values of $f(x)$.

We then generate n random samples as follows. The probability of choosing subinterval i is given by the probability

$$p_i = \frac{f(s_i)}{S}. \tag{11.51}$$

Note that the sum over all subintervals of p_i is normalized to unity. To choose a subinterval with the desired probability, we generate a random number r uniformly in the interval $[a, b]$ and determine the subinterval i that satisfies the inequality (11.22). Now that the subinterval has been chosen with the desired probability, we generate a random number x_i in the subinterval $[s_i, s_i + h]$ and compute the ratio $f(x_i)/p(x_i)$.

The estimate of the integral is given by the following considerations. The probability p_i in (11.51) is the probability of choosing the subinterval i, not the probability of choosing a value of x between x and $x + \Delta x$. The probability $p(x)\Delta x$ is p_i times the the probability of picking the particular value of x in subinterval i:

$$p(x_i)\Delta x = p_i \times \frac{\Delta x}{h}. \tag{11.52}$$

Hence, we have that

$$F_n = \frac{1}{n}\sum_{i=1}^{n} \frac{f(x_i)}{p(x_i)} = \frac{h}{n}\sum_{i=1}^{n} \frac{f(x_i)}{p_i}. \tag{11.53}$$

Problem 11.14. Apply the above method to estimate the integral of $f(x) = \sqrt{1 - x^2}$ in the unit interval.

11.8 Metropolis Algorithm

Another way of generating an arbitrary nonuniform probability distribution was introduced by Metropolis, Rosenbluth, Rosenbluth, Teller and Teller in 1953. The *Metropolis* algorithm is a special case of an importance sampling procedure in which certain possible sampling attempts are rejected (see Appendix 11C). The Metropolis method is useful for computing averages of the form

$$\langle f \rangle = \frac{\int p(x) f(x)\, dx}{\int p(x)\, dx}, \tag{11.54}$$

where $p(x)$ is an arbitrary probability distribution that need not be normalized. In Chapter **??** we will discuss the application of the Metropolis algorithm to problems in statistical mechanics.

For simplicity, we introduce the Metropolis algorithm in the context of estimating one-dimensional definite integrals. Suppose that we wish to use importance sampling to generate random variables according to an arbitrary probability density $p(x)$. The Metropolis algorithm produces a random walk of points $\{x_i\}$ whose asymptotic probability distribution approaches $p(x)$ after a large number of steps. The random walk is defined by specifying a *transition probability* $T(x_i \rightarrow x_j)$ from one value x_i to another value x_j such that the distribution of points $x_0, x_1, x_2, \ldots$ converges to $p(x)$. It can be shown that it is sufficient (but not necessary) to satisfy the "detailed balance" condition

$$p(x_i) T(x_i \rightarrow x_j) = p(x_j) T(x_j \rightarrow x_i). \tag{11.55}$$

The relation (11.55) does not specify $T(x_i \rightarrow x_j)$ uniquely. A simple choice of $T(x_i \rightarrow x_j)$ that is consistent with (11.55) is

$$T(x_i \rightarrow x_j) = \min \left[1, \frac{p(x_j)}{p(x_i)} \right]. \tag{11.56}$$

If the "walker" is at position x_i and we wish to generate x_{i+1}, we can implement this choice of $T(x_i \rightarrow x_j)$ by the following steps:

1. Choose a trial position $x_{\text{trial}} = x_i + \delta_i$, where δ_i is a random number in the interval $[-\delta, \delta]$.
2. Calculate $w = p(x_{\text{trial}})/p(x_i)$.
3. If $w \geq 1$, accept the change and let $x_{i+1} = x_{\text{trial}}$.
4. If $w < 1$, generate a random number r.
5. If $r \leq w$, accept the change and let $x_{i+1} = x_{\text{trial}}$.
6. If the trial change is not accepted, then let $x_{i+1} = x_i$.

It is necessary to sample many points of the random walk before the asymptotic probability distribution $p(x)$ is attained. How do we choose the maximum "step size" δ? If δ is too large, only a small percentage of trial steps will be accepted and the sampling of $p(x)$ will be inefficient. On the other hand, if δ is too small, a large percentage of trial steps will be accepted, but again the sampling of $p(x)$ will be inefficient. A rough criterion for the magnitude of δ is that approximately

one third to one half of the trial steps should be accepted. We also wish to choose the value of x_0 such that the distribution $\{x_i\}$ will approach the asymptotic distribution as quickly as possible. An obvious choice is to begin the random walk at a value of x at which $p(x)$ is a maximum.

Pseudocode that implements the Metropolis algorithm is given below.

```
double xtrial = x + (2*rnd.nextDouble() - 1.0)*delta;
double w = p(xtrial)/p(x);
if (rnd <= w)
{
   x = xtrial
   naccept++;       // number of acceptances
}
```

Problem 11.15. The Gaussian distribution

a. Write a program using the Metropolis algorithm to generate the Gaussian distribution, $p(x) = Ae^{-x^2/2}$. Is the value of the normalization constant A relevant? Determine the qualitative dependence of the acceptance ratio and the equilibration time on the maximum step size δ. One possible criterion for equilibrium is that $\langle x^2 \rangle \approx 1$. What is a reasonable choice for δ? How many trials are needed to reach equilibrium for your choice of δ?

b. Modify your program so that it plots the asymptotic probability distribution generated by the Metropolis algorithm.

c. Calculate the autocorrelation function $C(j)$ defined by

$$C(j) = \frac{\langle x_{i+j} x_i \rangle - \langle x_i \rangle^2}{\langle x_i^2 \rangle - \langle x_i \rangle^2}, \tag{11.57}$$

where $\langle \ldots \rangle$ indicates an average over the random walk. What is the value of $C(j = 0)$? What would be the value of $C(j \neq 0)$ if x_i were completely random? Calculate $C(j)$ for different values of j and determine the value of j for which $C(j)$ is essentially zero.

Problem 11.16. Application of the Metropolis algorithm

a. Although the Metropolis algorithm is not the most efficient method in this case, write a program to estimate the average

$$\langle x \rangle = \frac{\int_0^\infty x e^{-x}\, dx}{\int_0^\infty e^{-x}\, dx}, \tag{11.58}$$

with $p(x) = Ae^{-x}$ for $x \geq 0$ and $p(x) = 0$ for $x < 0$. Incorporate into the program a computation of the histogram $H(x)$ showing the fraction of points in the random walk in the region x to $x + \Delta x$, with $\Delta x = 0.2$. Begin with $n = 1000$ and maximum step size $\delta = 1$. Allow the system to equilibrate for 200 steps before computing averages. Is the integrand sampled uniformly? If not, what is the approximate region of x where the integrand is sampled more often?

b. Calculate analytically the exact value of $\langle x \rangle$. How do your Monte Carlo results compare with the exact value for $n = 100$ and $n = 1000$ with $\delta = 0.1, 1$, and 10? Estimate the standard error of the mean. Does this error give a reasonable estimate of the error? If not, why?

c. In part (b) you should have found that the estimated error is much smaller than the actual error. The reason is that the $\{x_i\}$ are not statistically independent. The Metropolis algorithm produces a random walk whose points are correlated with each other over short times (measured in the number of Monte Carlo steps). The correlation of the points decays exponentially with time. If τ is the characteristic time for this decay, then only points separated by approximately 2 to 3τ can be considered statistically independent. Rerun your program with the data grouped into 20 sets of 50 points each and 10 sets of 100 points each. If the sets of 50 points each are statistically independent (that is, if τ is significantly smaller than 50), then your estimate of the error for the two groupings should be approximately the same.

Appendix 11A: Error Estimates for Numerical Integration

We derive the dependence of the truncation error estimates on the number of intervals for the numerical integration methods considered in Sections 11.1 and 11.3. These estimates are based on the assumed adequacy of the Taylor series expansion of the integrand $f(x)$:

$$f(x) = f(x_i) + f'(x_i)(x - x_i) + \frac{1}{2}f''(x_i)(x - x_i)^2 + \dots , \tag{11.59}$$

and the integration of (11.1) in the interval $x_i \le x \le x_{i+1}$:

$$\int_{x_i}^{x_{i+1}} f(x)\,dx = f(x_i)\Delta x + \frac{1}{2}f'(x_i)(\Delta x)^2 + \frac{1}{6}f''(x_i)(\Delta x)^3 + \dots \tag{11.60}$$

We first estimate the error associated with the rectangular approximation with $f(x)$ evaluated at the left side of each interval. The error Δ_i in the interval $[x_i, x_{i+1}]$ is the difference between (11.60) and the estimate $f(x_i)\Delta x$:

$$\Delta_i = \left[\int_{x_i}^{x_{i+1}} f(x)\,dx\right] - f(x_i)\Delta x \approx \frac{1}{2}f'(x_i)(\Delta x)^2. \tag{11.61}$$

We see that to leading order in Δx, the error in each interval is order $(\Delta x)^2$. Because there are a total of n intervals and $\Delta x = (b - a)/n$, the total error associated with the rectangular approximation is $n\Delta_i \sim n(\Delta x)^2 \sim n^{-1}$.

The estimated error associated with the trapezoidal approximation can be found in the same way. The error in the interval $[x_i, x_{i+1}]$ is the difference between the exact integral and the estimate, $\frac{1}{2}[f(x_i) + f(x_{i+1})]\Delta x$:

$$\Delta_i = \left[\int_{x_i}^{x_{i+1}} f(x)\,dx\right] - \frac{1}{2}[f(x_i) + f(x_{i+1})]\Delta x. \tag{11.62}$$

If we use (11.60) to estimate the integral and (11.59) to estimate $f(x_{i+1})$ in (11.62), we find that the term proportional to f' cancels and that the error associated with one interval is order $(\Delta x)^3$. Hence, the total error in the interval $[a, b]$ associated with the trapezoidal approximation is order n^{-2}.

Because Simpson's rule is based on fitting $f(x)$ in the interval $[x_{i-1}, x_{i+1}]$ to a parabola, error terms proportional to f'' cancel. We might expect that error terms of order $f'''(x_i)(\Delta x)^4$ contribute, but these terms cancel by virtue of their symmetry. Hence the $(\Delta x)^4$ term of the Taylor expansion of $f(x)$ is adequately represented by Simpson's rule. If we retain the $(\Delta x)^4$ term in the Taylor series of $f(x)$, we find that the error in the interval $[x_i, x_{i+1}]$ is of order $f''''(x_i)(\Delta x)^5$ and that the total error in the interval $[a, b]$ associated with Simpson's rule is $O(n^{-4})$.

The error estimates can be extended to two dimensions in a similar manner. The two-dimensional integral of $f(x, y)$ is the volume under the surface determined by $f(x, y)$. In the "rectangular" approximation, the integral is written as a sum of the volumes of parallelograms with cross sectional area $\Delta x \Delta y$ and a height determined by $f(x, y)$ at one corner. To determine the error, we expand $f(x, y)$ in a Taylor series

$$f(x, y) = f(x_i, y_i) + \frac{\partial f(x_i, y_i)}{\partial x}(x - x_i) + \frac{\partial f(x_i, y_i)}{\partial y}(y - y_i) + \dots , \tag{11.63}$$

and write the error as

$$\Delta_i = \left[\int\int f(x, y)\, dxdy\right] - f(x_i, y_i)\Delta x \Delta y. \tag{11.64}$$

If we substitute (11.63) into (11.64) and integrate each term, we find that the term proportional to f cancels and the integral of $(x - x_i)\, dx$ yields $\frac{1}{2}(\Delta x)^2$. The integral of this term with respect to dy gives another factor of Δy. The integral of the term proportional to $(y - y_i)$ yields a similar contribution. Because Δy also is order Δx, the error associated with the intervals $[x_i, x_{i+1}]$ and $[y_i, y_{i+1}]$ is to leading order in Δx:

$$\Delta_i \approx \frac{1}{2}[f'_x(x_i, y_i) + f'_y(x_i, y_i)](\Delta x)^3. \tag{11.65}$$

We see that the error associated with one parallelogram is order $(\Delta x)^3$. Because there are n parallelograms, the total error is order $n(\Delta x)^3$. However in two dimensions, $n = A/(\Delta x)^2$, and hence the total error is order $n^{-1/2}$. In contrast, the total error in one dimension is order n^{-1}, as we saw earlier.

The corresponding error estimates for the two-dimensional generalizations of the trapezoidal approximation and Simpson's rule are order n^{-1} and n^{-2} respectively. In general, if the error goes as order n^{-a} in one dimension, then the error in d dimensions goes as $n^{-a/d}$. In contrast, Monte Carlo errors vary as order $n^{-1/2}$ independent of d. Hence for large enough d, Monte Carlo integration methods will lead to smaller errors for the same choice of n.

Appendix 11B: The Standard Deviation of the Mean

In Section 11.4 we gave empirical reasons for the claim that the error associated with a single measurement consisting of n trials equals $\sigma/\sqrt{n}$, where σ is the standard deviation in a single measurement. We now present an analytical derivation of this relation.

The quantity of experimental interest is denoted as x. Consider m sets of measurements each with n trials for a total of mn trials. We use the index α to denote a particular measurement

and the index i to designate the ith trial within a measurement. We denote $x_{\alpha,i}$ as trial i in the measurement α. The value of a measurement is given by

$$M_\alpha = \frac{1}{n}\sum_{i=1}^{n} x_{\alpha,i}. \tag{11.66}$$

The mean $\overline{M}$ of the *total* mn individual trials is given by

$$\overline{M} = \frac{1}{m}\sum_{\alpha=1}^{m} M_\alpha = \frac{1}{nm}\sum_{\alpha=1}^{m}\sum_{i=1}^{n} x_{\alpha,i}. \tag{11.67}$$

The difference between measurement α and the mean of all the measurements is given by

$$e_\alpha = M_\alpha - \overline{M}. \tag{11.68}$$

We can write the variance of the means as

$${\sigma_m}^2 = \frac{1}{m}\sum_{\alpha=1}^{m} {e_\alpha}^2. \tag{11.69}$$

We now wish to relate σ_m to the variance of the individual trials. The discrepancy $d_{\alpha,i}$ between an individual sample $x_{\alpha,i}$ and the mean is given by

$$d_{\alpha,i} = x_{\alpha,i} - \overline{M}. \tag{11.70}$$

Hence, the variance σ^2 of the nm individual trials is

$$\sigma^2 = \frac{1}{mn}\sum_{\alpha=1}^{m}\sum_{i=1}^{n} {d_{\alpha,i}}^2. \tag{11.71}$$

We write

$$e_\alpha = M_\alpha - \overline{M} = \tfrac{1}{n}\textstyle\sum_{i=1}^{n}\left(x_{\alpha,i} - \overline{M}\right) \tag{11.72}$$

$$= \tfrac{1}{n}\textstyle\sum_{i=1}^{n} d_{\alpha,i}. \tag{11.73}$$

If we substitute (11.73) into (11.69), we find

$${\sigma_m}^2 = \frac{1}{m}\sum_{\alpha=1}^{n}\left(\frac{1}{n}\sum_{i=1}^{n} d_{\alpha,i}\right)\left(\frac{1}{n}\sum_{j=1}^{n} d_{\alpha,j}\right). \tag{11.74}$$

The sum in (11.74) over trials i and j in set α contains two kinds of terms—those with $i = j$ and those with $i \neq j$. We expect that $d_{\alpha,i}$ and $d_{\alpha,j}$ are independent and equally positive or negative on the average. Hence in the limit of a large number of measurements, we expect that only the terms with $i = j$ in (11.74) will survive, and we write

$${\sigma_m}^2 = \frac{1}{mn^2}\sum_{\alpha=1}^{n}\sum_{i=1}^{n} {d_{\alpha,i}}^2. \tag{11.75}$$

If we combine (11.75) with (11.71), we arrive at the desired result

$${\sigma_m}^2 = \frac{\sigma^2}{n}. \tag{11.76}$$

Appendix 11C: The Acceptance-Rejection Method

Although the inverse transform method discussed in Section 11.5 can in principle be used to generate any desired probability distribution, in practice the method is limited to functions for which the equation, $r = P(x)$, can be solved analytically for x or by simple numerical approximation. Another method for generating nonuniform probability distributions is the *acceptance-rejection* method due to von Neumann.

Suppose that $p(x)$ is a (normalized) probability density function that we wish to generate. For simplicity, we assume $p(x)$ is nonzero in the unit interval. Consider a positive definite *comparison function* $w(x)$ such that $w(x) > p(x)$ in the entire range of interest. A simple although not generally optimum choice of w is a constant greater than the maximum value of $p(x)$. Because the area under the curve $p(x)$ in the range x to $x + \Delta x$ is the probability of generating x in that range, we can follow a procedure similar to that used in the hit or miss method. Generate two numbers at random to define the location of a point in two dimensions which is distributed uniformly in the area under the comparison function $w(x)$. If this point is outside the area under $p(x)$, the point is rejected; if it lies inside the area, we accept it. This procedure implies that the accepted points are uniform in the area under the curve $p(x)$ and that their x values are distributed according to $p(x)$.

One procedure for generating a uniform random point (x, y) under the comparison function $w(x)$ is as follows.

1. Choose a form of $w(x)$. One choice would be to choose $w(x)$ such that the values of x distributed according to $w(x)$ can be generated by the inverse transform method. Let the total area under the curve $w(x)$ be equal to A.

2. Generate a uniform random number in the interval $[0, A]$ and use it to obtain a corresponding value of x distributed according to $w(x)$.

3. For the value of x generated in step (2), generate a uniform random number y in the interval $[0, w(x)]$. The point (x, y) is uniformly distributed in the area under the comparison function $w(x)$. If $y \leq p(x)$, then accept x as a random number distributed according to $p(x)$.

Repeat steps (2) and (3) many times.

Note that the acceptance-rejection method is efficient only if the comparison function $w(x)$ is close to $p(x)$ over the entire range of interest.

References and Suggestions for Further Reading

Forman S. Acton, *Numerical Methods That Work*, Harper & Row (1970); corrected edition, Mathematical Association of America (1990). A delightful book on numerical methods.

Isabel Beichl and Francis Sullivan, "The Importance of Importance Sampling," *Computing in Science and Engineering* **1**(#2), 71–73 (1999).

Steven E. Koonin and Dawn C. Meredith, *Computational Physics*, Addison-Wesley (1990). Chapter 8 covers much of the same material on Monte Carlo methods as discussed in this chapter.

Malvin H. Kalos and Paula A. Whitlock, *Monte Carlo Methods, Vol. 1: Basics,* John Wiley & Sons (1986). The authors are well known experts on Monte Carlo methods.

William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, *Numerical Recipes*, second edition, Cambridge University Press (1992).

Reuven Y. Rubinstein, *Simulation and the Monte Carlo Method*, John Wiley & Sons (1981). An advanced, but clearly written treatment of Monte Carlo methods.

I. M. Sobol, *The Monte Carlo Method*, Mir Publishing (1975). A very readable short text with excellent sections on nonuniform probability densities and the neutron transport problem.