11
Monte Carlo Applications

Now that we have an idea of how to use the computer to generate a series of pseudo random numbers, we must build some confidence that using these numbers in a calculation is a way of incorporating the element of chance into a simulation. We do this first by simulating a random walk and then an atom decaying spontaneously. After that, we show how knowing the statistics of random numbers leads to the best way to evaluate multidimensional integrals.

11.1
A Random Walk (Problem)

There are many physical processes, such as Brownian motion and electron transport through metals, in which a particle appears to move randomly. For example, consider a perfume molecule released in the middle of a classroom. It collides randomly with other molecules in the air and eventually reaches the instructor's nose. The **problem** is to determine how many collisions, on average, the molecule must make to travel a radial distance of R, if it travels a step length $r_{\rm rms}$ on the average (a *root-mean-square* average) between collisions.

11.1.1
Random Walk Simulation

There are a number of ways to simulate a random walk, and (surprise, surprise) different assumptions give different physics. We will present the simplest approach for a 2D walk, with a minimum of theory, and end up with a model for normal diffusion. *The research literature is full of discussions of various versions of this problem. For example, Brownian motion corresponds to the limit in which the individual step lengths approach zero with no time delay between steps. Additional refinements include collisions within a moving medium (*abnormal diffusion*), including the velocities of the particles, or even pausing between steps. Models such as these are discussed in Chap. 20.*

Computationyal Physics. Problem Solving with Computers (2nd edn).
Rubin H. Landau, Manuel José Páez, Cristian C. Bordeianu
Copyright © 2007 WILEY-VCH Verlag GmbH & Co. KGaA, Weinheim
ISBN: 978-3-527-40626-5

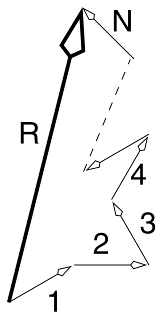

Fig. 11.1 Some of the N steps in a random walk that end up a distance R from the origin. Notice how the Δx's for each step add algebraically.

In a random-walk simulation, such as that in Fig. 11.1 from the code `Walk.java` on the Instructor's CD, an artificial *walker* takes many steps, usually with the *direction* of each step *independent* from the direction of the previous one (Fig. 11.1). For our model, we start at the origin and take N steps in the x–y plane of *lengths* (not coordinates)

$$(\Delta x_1, \Delta y_1), (\Delta x_2, \Delta y_2), (\Delta x_3, \Delta y_3), \dots, (\Delta x_n, \Delta y_N) \tag{11.1}$$

The radial distance R from the starting point traveled after N steps is

$$\begin{aligned} R^2 &= (\Delta x_1 + \Delta x_2 + \cdots + \Delta x_N)^2 + (\Delta y_1 + \Delta y_2 + \cdots + \boldsymbol{\Delta} y_N)^2 \\ &= \Delta x_1^2 + \Delta x_2^2 + \cdots + \Delta x_N^2 + 2\Delta x_1 \Delta x_2 + 2\Delta x_1 \Delta x_3 + 2\Delta x_2 \Delta x_1 + \cdots \\ &\quad + (x \rightarrow y) \end{aligned} \tag{11.2}$$

Equation (11.2) is valid for any walk. If it is random, the particle is equally likely to in any direction in each step. *On average*, for a large number of random steps, all the cross terms in (11.2) will vanish and we will be left with

$$\begin{aligned} R^2 &\simeq \Delta x_1^2 + \Delta x_2^2 + \cdots + \Delta x_N^2 + \Delta y_1^2 + \Delta y_2^2 + \cdots + \Delta y_N^2 = N\langle r^2 \rangle \\ \Rightarrow \qquad R &\simeq \sqrt{N} r_{\text{rms}} \end{aligned} \tag{11.3}$$

Here r_{rms} is the average (*root-mean-squared*) step size.

In summary, (11.3) indicates that for a walk of N steps covering a total distance of $N r_{\text{rms}}$, the walk ends up, on the average, a radial distance $\sqrt{N} r_{\text{rms}}$ from the starting point. For large N this is significantly less than $N r_{\text{rms}}$, but is *not* zero (which is the average of the *displacement* vector). In our experience, practical simulations agree with this theory, but rarely perfectly, with the level of agreement depending upon how the averages are taken, and just how the randomness is built into each step.

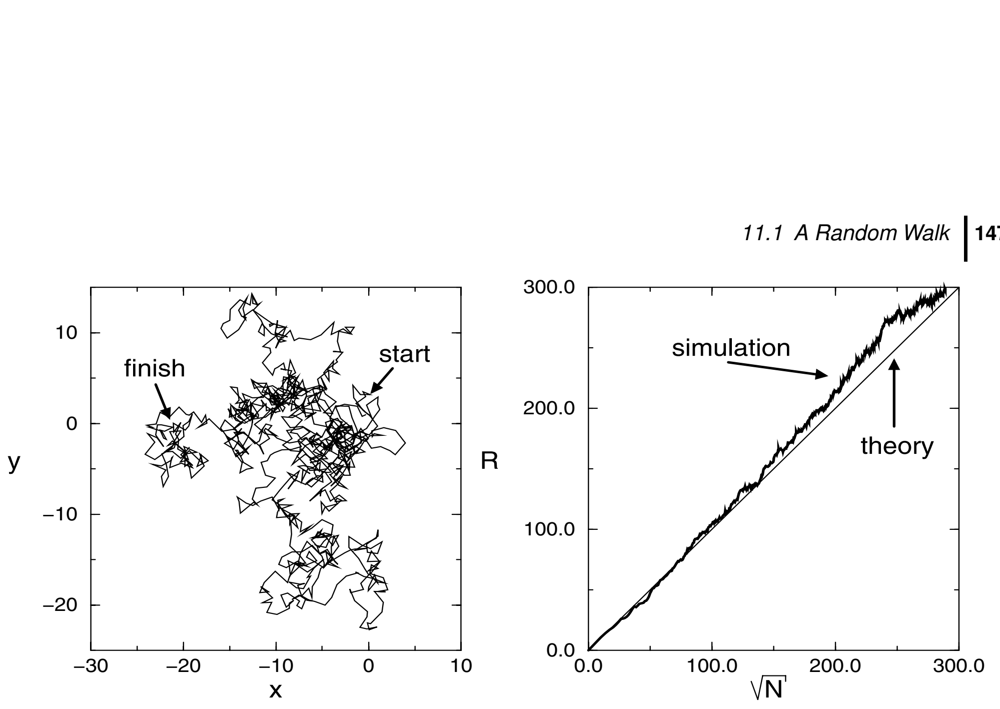

Fig. 11.2 *Left*: A computer simulation of a random walk. *Right*: the distance covered in a simulated random walk of N steps compared to the theoretical prediction (11.3).

11.1.2
Implementation: Random Walk

The program `Walk.java` on the instructor's CD contains our simulation of a random walk. It's key element is random values for the x and y components of each step,

```
x += (randnum.nextDouble() -0.5);  y += (randnum.nextDouble() -0.5);
r[i] += Math.sqrt((x*x)+(y*y));                  // radius
```

where we leave off the scaling factor that normalizes each step to length 1. When using your computer to simulate a random walk, you should only expect to obtain (11.3) as the average displacement after many trials, not necessarily as the answer for each trial. You may get different answers depending on how you take your random steps (Fig. 11.2). Start at the origin and take a 2D random walk with your computer.

1. To increase the amount of randomness, independently choose random values for $\Delta x'$ and $\Delta y'$ in the range $[-1, 1]$. Then normalize them so that the step is of unit length:

$$\Delta x = \frac{1}{L}\Delta x' \qquad \Delta y = \frac{1}{L}\Delta y' \qquad L = \sqrt{\Delta x'^2 + \Delta y'^2}$$

2. Use a plotting program to draw maps of several independent random walks, each of 1000 steps. Comment on whether these look like what you expect of a random walk.

3. If you have your walker taking N steps in a single trial, then conduct a total of $K \simeq \sqrt{N}$ trials. Each trial should have N steps and start with a different seed.

4. Calculate the mean-square distance R^2 for each trial, and then take the average of R^2 for all your K trials:

$$\langle R^2(N) \rangle = \frac{1}{K} \sum_{k=1}^{K} R^2_{(k)}(N)$$

5. Plot the root mean-square distance $R_{\text{rms}} = \sqrt{\langle R^2(N) \rangle}$ as a function of $\sqrt{N}$. Values of N should start with a small number where $R \simeq \sqrt{N}$ is not expected to be accurate, and end at a quite large value, where 2–3 places of accuracy should be expected on the average.

11.2 Radioactive Decay (Problem)

We have already encountered spontaneous radioactive decay in Chap. 8, where we fit an exponential function to a decay spectrum. Your **problem** now is to simulate how a small number of radioactive particles decay. In particular, you are to determine when radioactive decay looks exponential and when it looks *stochastic* (that is, determined by chance). Because the exponential decay law is only a large-number approximation to the natural process, our simulation should be closer to nature than the exponential decay law (Fig. 11.3). In fact, if you go to the CD and "listen" to the output of the decay simulation code, what you hear sounds very much like a Geiger counter: a convincing demonstration of how realistic the simulation is.

Spontaneous decay is a natural process in which a particle, with no external stimulation, and at one instant in time, decays into other particles. Because the exact moment when any one particle decays is random, it does not matter how long the particle has been around or what is happening to the other particles. In other words, the probability $\mathcal{P}$ of any one particle decaying per unit time is a constant, and when that particle decays, it is gone forever. Of course, as the number of particles decreases with time, so will the number of decays. Nonetheless, the probability of any one particle decaying in some time interval is always the same constant as long as the particle still exists.

11.2.1 Discrete Decay (Model)

Imagine having a sample of $N(t)$ radioactive nuclei at time t (Fig. 11.3, left). Let ΔN be the number of particles that decay in some small time interval Δt. We convert the statement "the probability $\mathcal{P}$ of any one particle decaying per unit time is a constant" into an equation by noting that the decay probability per particle, $\Delta N/N$, is proportional to the length of the time interval over

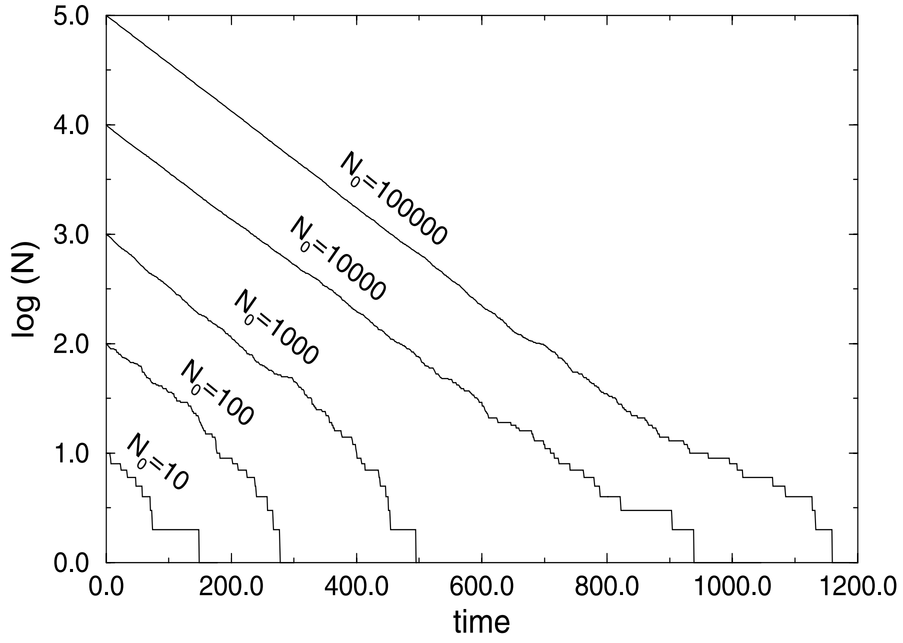

Fig. 11.3 Semilog plots of the results from several decay simulations. Notice how the decay appears exponential (like a straight line) when the number of nuclei is large, but stochastic for $\log N \leq 2.0$.

which we observe the particle

$$\mathcal{P} = \frac{\Delta N(t)}{N(t)} = -\lambda\,\Delta t \qquad \Rightarrow \qquad \frac{\Delta N(t)}{N(t)\Delta t} = -\lambda \tag{11.4}$$

where λ is a constant. Sure enough, (11.4) says that the probability of any one particle decaying per unit time is a constant.

Equation (11.4) is a *finite-difference equation* in which $\Delta N(t)$ and Δt are experimental observables. Although it cannot be integrated the way a differential equation can, it can be solved numerically or algebraically. Because the decay process is random, we cannot predict an exact value for $\Delta N(t)$. Instead, we may think of $\Delta N(t)$ as the average number of decays when observations are made of many identical systems of N radioactive particles.

We convert (11.4) into a finite-difference equation for the decay rate by multiplying both sides by $N(t)$:

$$\frac{\Delta N(t)}{\Delta t} = -\lambda N(t) \tag{11.5}$$

The absolute decay rate $\Delta N(t)/\Delta t$ is called the *activity*, and because it is proportional to the number of particles present, it too has an exponential-like decay in time. However, the dynamics of the simulation is still a random process, so eventually the activity too becomes stochastic. Actually, because the activity $\Delta N(t)/\Delta t$ is proportional to the difference in random numbers, its stochastic nature become evident before that of $N(t)$.

11.2.2 Continuous Decay (Model)

When the number of particles $N \to \infty$ and the observation time interval approaches zero, an approximate form of the radioactive decay law (11.5) results:

$$\frac{\Delta N(t)}{\Delta t} \longrightarrow \frac{dN(t)}{dt} = -\lambda N(t) \tag{11.6}$$

This can be integrated to obtain the exponential decay law for the number and for the activity:

$$N(t) = N(0)e^{-\lambda t} = N(0)e^{-t/\tau} \tag{11.7}$$

$$\frac{dN}{dt}(t) = -\lambda N(0)e^{-\lambda t} = \frac{dN}{dt}(0)e^{-\lambda t} \tag{11.8}$$

In this limit we get exponential decay. We identify the decay rate λ with the inverse of the lifetime:

$$\lambda = \frac{1}{\tau} \tag{11.9}$$

We see from its derivation that exponential decay is a good description of nature only *on the average*, and only for a large number of particles. The basic law of nature (11.4) is always valid, but, as we will see in the simulation, (11.8) becomes less and less accurate as the number of particles gets smaller and smaller.

11.2.3 Decay Simulation

A program for simulating radioactive decay is surprisingly simple, but not without its subtleties. It increases time in discrete steps of Δt, and at each time counts how many nuclei have decayed during the last Δt. The simulation quits when there are no nuclei left. Such being the case, we have an outer loop over the time steps Δt, and an inner loop over the nuclei that are remaining at the present time. The pseudocode for this simulation is simple:

```
input N, lambda  t=0  while N > 0
    DeltaN = 0
    for i = 1...N
        if (r_i < lambda) DeltaN = DeltaN + 1
    end for
    t = t +1
    N = N − DeltaN
    Output t , DeltaN , N
end while
```

At some point in writing the simulation program, we set the scale, or units, in which is measured. Since the decay rate parameter $\lambda = 1/\tau$, where τ is the lifetime of the nucleus, picking a value of λ essentially sets the time scale. Since the simulation compares λ to a random number, and since random numbers are usually in the range $0 \le r_i \le 1$, it is convenient to pick time units for which $0 \le \lambda \le 1$ (for example, $\lambda \simeq 0.3$ is a good place to start). This means that when the do loop counts time with $\Delta t = 1$ units, the time scale has been set as $1/\lambda$.

As an example, let us imagine a nucleus with a lifetime $\tau = 10$ s. This corresponds to a decay parameter $\lambda = 1/\tau = 0.1$/s. So if we use $\lambda = 0.1$ in the simulation, each time step would correspond to 1 s. Likewise, if we have a nucleus with a lifetime $\tau = 10 \times 10^{-6}$ s, then this corresponds to a decay parameter $\lambda = 1/\tau = 0.1/10^{-6}$ s. So if we use $\lambda = 0.1$ in the simulation, each time step would correspond to 10^{-6} of a second. The actual value of the decay rate $\Delta N/\Delta t$ in particles per second would then be $\Delta N/10^{-6}$. However, unless you plan to compare your simulation to experimental data, you do not have to worry about the scale for time and can output ΔN as the decay rate (it is actually the physics behind the slopes and relative magnitudes of the graphs that we want to show).

11.3 Decay Implementation and Visualization

Write your own program to simulate radioactive decay, using our sample program in Listing 11.1 as a guide (it is simple). You should obtain results like those on the right of Fig. 11.3.

Listing 11.1: `Decay.java` simulates spontaneous decay having decay whenever a random number is smaller than a normalized decay rate.

```java
// Decay.java: Spontaneous decay simulation

import java.io.*;
import java.util.*;

public class Decay {
  static double lambda = 0.01;                                  // Decay constant
  static int max = 1000, time_max = 500, seed = 68111;          // Params

  public static void main(String[] argv)
                       throws IOException, FileNotFoundException {
```

```
    int atom, time, number, nloop;
    double decay;

    PrintWriter w =
        new PrintWriter(new FileOutputStream("decay.dat"), true);
    number = nloop = max;                                  // Initial value
    Random r = new Random(seed);                 // Seed random generator
                                                            // Time loop
    for ( time = 0;  time <= time_max;  time++ ) {
                                                            // Atom loop
      for ( atom = 1;  atom <= number;  atom++ )  {
        decay = r.nextDouble();
        if (decay  <  lambda) nloop--;
      }                                                  // An atom decays
      number = nloop;
      w.println( " " + time + "  " + (double)number/max);
    }
   System.out.println("data stored in decay.dat");
  }
}
```

1. Plot the logarithm of the number left $\ln N(t)$ and the logarithm of the decay rate $\ln(\Delta N(t))$ versus time. Note that the simulation measures time in steps of Δt (generation number).
2. Check that you obtain what looks like exponential decay when you start with large values for $N(0)$, and that for small $N(0)$ you get a stochastic process (the large $N(0)$ simulation is also stochastic, it just does not look it).
3. Create two plots, one showing that the slopes of plots of $N(t)$ versus t are *independent* of $N(0)$, and the another showing that the slope is proportional to λ.
4. Create a plot to show that, within the expected stochastic variations, $\ln N(t)$ and $\ln(\Delta N(t))$ are proportional.
5. Explain in your own words how a process that is spontaneous and random at its very heart leads to exponential decay.
6. How does your simulation show that the decay is exponential and not a power law like $N = \beta t^{-\alpha}$?

11.4
Integration by Stone Throwing (Problem)

Imagine yourself as a farmer walking to your furthermost field to add chemicals to a pond having an algae explosion. You get there, only to read the in-

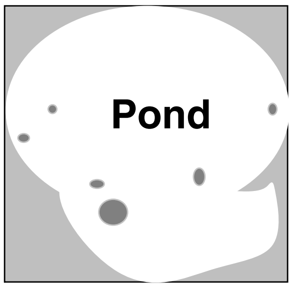

Fig. 11.4 Throwing stones in a pond as a technique for measuring its area. There is a tutorial of this on the CD where you can see the actual "splashes" (the dark spots) used in an integration.

structions and discover that you need to know the area of the pond to get the correct concentration. Your **problem** is to measure the area of this irregularly shaped pond with just the materials at hand [17].

11.5 Integration by Rejection (Theory)

It is hard to believe that Monte Carlo techniques could be used to evaluate integrals. After all, we do not want to gamble on their values! While it is true that other methods are preferable for single and double integrals, when the number of integrations required gets large, Monte Carlo techniques are the best!

For our pond problem, we will use the *sampling* technique (Fig. 11.4):

1. Walk off a box that completely encloses the pond and remove any pebbles within the box.
2. Measure the lengths of the sides in natural units like *feet*. This tells you the area of the enclosing box A_{box}.
3. Grab a bunch of pebbles and throw them up in the air in random directions.
4. Count the number of splashes in the pond N_{pond} and the number of pebbles lying on the ground within your box N_{box}.
5. Assuming that you threw the pebbles uniformly and randomly, the number of pebbles falling into the pond should be proportional to the

area of the pond A_{pond}. You determine that area from the simple ratio

$$\begin{aligned} \frac{N_{\text{pond}}}{N_{\text{pond}} + N_{\text{box}}} &= \frac{A_{\text{pond}}}{A_{\text{box}}} \\ \Rightarrow \qquad A_{\text{pond}} &= \frac{N_{\text{pond}}}{N_{\text{pond}} + N_{\text{box}}} A_{\text{box}} \end{aligned} \tag{11.10}$$

11.5.1 Stone Throwing Implementation

Use sampling (Fig. 11.4) to perform a 2D integration and thereby determine π:

1. Imagine a circular pond centered at the origin and enclosed in a square of side 2.
2. We know the analytic result
$$\oint dA = \pi \tag{11.11}$$
3. Generate a sequence of random numbers $\{r_i\}$ in $[-1, 1]$.
4. For $i = 1$ to N, pick $(x_i, y_i) = (r_{2i-1}, r_{2i})$.
5. If $x_i^2 + y_i^2 < 1$, let $N_{\text{pond}} = N_{\text{pond}} + 1$, else let $N_{\text{box}} = N_{\text{box}} + 1$.
6. Use (11.10) to calculate the area and in this way π.

Try increasing N until you get π to three significant figures (we do not ask much; that is only slide-rule accuracy).

11.5.2 Integration by Mean Value (Math)

The standard Monte Carlo technique for integration is based on the *mean-value theorem* (presumably familiar from elementary calculus):

$$I = \int_a^b dx f(x) = (b - a)\langle f \rangle \tag{11.12}$$

The theorem states the obvious if you think of integrals as areas: the value of the integral of some function $f(x)$ between a and b equals the length of the interval $(b - a)$ times the average value of the function over that interval $\langle f \rangle$ (Fig. 11.5). The integration algorithm uses Monte Carlo techniques to evaluate the mean in (11.12). With a sequence x_i of N uniform random numbers in $[a, b]$, we want to determine the *sample mean* by *sampling* the function $f(x)$ at these points:

$$\langle f \rangle \simeq \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{11.13}$$

This gives us the very simple integration rule:

$$\int_a^b dx\, f(x) \simeq (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i) = (b-a)\langle f \rangle \tag{11.14}$$

Equation (11.14) looks much like our standard algorithm for integration (5.3), with the "points" x_i chosen randomly and with constant weights $w_i = (b-a)/N$. Because no attempt has been made to get the best answer for a given value of N, this is by no means an optimized way to evaluate integrals; but you will admit it is simple. If we let the number of samples of $f(x)$ approach infinity $N \to \infty$, or we keep the number of samples finite and take the average of infinitely many runs, the laws of statistics assure us that, barring roundoff errors, (11.14) approaches the correct answer.

For those readers who are somewhat familiar with statistics, we remind you that the uncertainty in the value obtained for the integral I after N samples of $f(x)$ is measured by the standard deviation σ_I. The standard deviation of the integrand f in the sampling is an intrinsic property of the function $f(x)$, that is, something we do not change by taking more samples. For normal distributions, the two are related by

$$\sigma_I \approx \frac{1}{\sqrt{N}}\sigma_f \tag{11.15}$$

This means that for large N, the error in the value of the integral always decreases as $1/\sqrt{N}$.

11.6
High-Dimensional Integration (Problem)

Let us say that we want to calculate some properties of a small atom such as magnesium with 12 electrons. To do that, we need to integrate some function over the three coordinates of each electron. This amounts to a $3 \times 12 = 36$-dimensional integral. If we use 64 points for each integration, this requires some $64^{36} \simeq 10^{65}$ evaluations of the integrand. If the computer were fast and could evaluate the integrand a million times per second, this would take some 10^{59} seconds, which is significantly longer than the age of the universe ($\sim 10^{17}$ s).

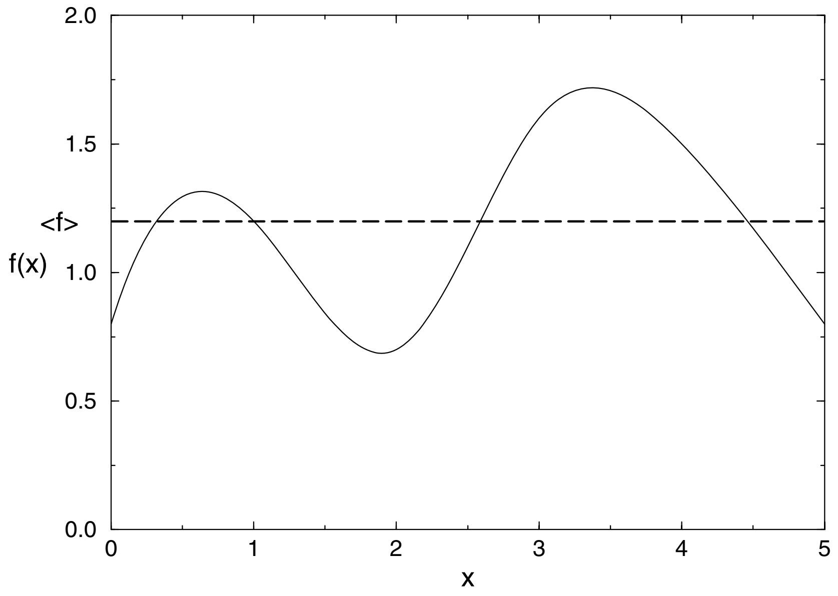

Fig. 11.5 The area under the curve $f(x)$ is the same as that under the dashed line $y = \langle f \rangle$.

Your **problem** is to find a way to perform multidimensional integrations so that you are still alive to savor the answers. Specifically, evaluate the 10D integral

$$I = \int_0^1 dx_1 \int_0^1 dx_2 \cdots \int_0^1 dx_{10}\,(x_1 + x_2 + \cdots + x_{10})^2 \tag{11.16}$$

Check your numerical answer against the analytic one, $\frac{155}{6}$.

11.6.1 Multidimensional Monte Carlo

It is easy to generalize mean-value integration to many dimensions by picking random points in a multidimensional space. For example,

$$\int_a^b dx \int_c^d dy\, f(x,y) \simeq (b-a)(d-c)\frac{1}{N}\sum_i^N f(\mathbf{x}_i) = (b-a)(d-c)\langle f \rangle \tag{11.17}$$

11.6.2 Error in Multidimensional Integration (Assessment)

When we perform a multidimensional integration, the error in the Monte Carlo technique, being statistical, decreases as $1/\sqrt{N}$. This is valid even if the N points are distributed over D dimensions. In contrast, when we use these same N points to perform a D-dimensional integration as D one-dimensional integrals, we use N/D points for each integration. For fixed N, this means that the number of points used for each integration decreases as the number of dimensions D increases, and so the error in each integration increases with D. Furthermore, the total error will be approximately N times the error in each integral. If we put these trends together and look at a particular integration rule, we would find that at a value of $D \simeq 3$–4 the error in Monte Carlo integration is similar to that of conventional schemes. For larger values of D, the Monte Carlo method is more accurate!

11.6.3 Implementation: 10D Monte Carlo Integration

Use a built-in random-number generator to perform the 10-dimensional Monte Carlo integration in (11.16). Our program `Int10d.java` is available on the instructor's CD.

1. Conduct 16 trials and take the average as your answer.
2. Try sample sizes of $N = 2, 4, 8, \ldots, 8192$.
3. Plot the absolute value of the error versus $1/\sqrt{N}$ and try to identify linear behavior.

11.7 Integrating Rapidly Varying Functions (Problem) ⊙

It is common in many physical applications to integrate a function with an approximately Gaussian dependence on x. The rapid falloff of the integrand means that our Monte Carlo integration technique would require an incredibly large number of integrations points to obtain even modest accuracy. Your **problem** is to make Monte Carlo integration more efficient for rapidly varying integrands.

11.7.1 Variance Reduction ⊙ (Method)

If the function being integrated never differs much from its average value, then the standard Monte Carlo mean-value method (11.14) should work well with a not too ridiculously large number of points. Yet for a function with a large *variance* (i.e., one that is not "flat"), many of the random evaluations of the function may occur where the function makes a slight contribution to the integral; this is, basically, a waste of time. The method can be improved by mapping the function f into a function g that has a smaller variance over the interval. We indicate two methods here and refer you to the References (at the end of this book) for more details.

The first method is a *variance reduction* or *subtraction technique* in which we devise a flatter function on which to apply the Monte Carlo technique. Suppose we construct a function $g(x)$ with the following properties on $[a, b]$:

$$|f(x) - g(x)| \le \epsilon \qquad \int_a^b dx\, g(x) = J \tag{11.18}$$

We now evaluate the integral of $f(x) - g(x)$ and add the result to J to obtain the required integral

$$\int_a^b dx\, f(x) = \int_a^b dx\, \{f(x) - g(x)\} + J \tag{11.19}$$

If we are clever enough to find a simple $g(x)$ that makes the variance of $f(x) - g(x)$ less than that of $f(x)$, and that we can integrate analytically, we obtain more accurate answers in less time.

11.7.2 Importance Sampling (Method) ⊙

A second method to improve Monte Carlo integration is called *importance sampling* because it lets us sample the integrand in the most important regions. It derives from expressing the integral in the form

$$I = \int_a^b dx\, f(x) = \int_a^b dx\, w(x) \frac{f(x)}{w(x)} \tag{11.20}$$

If we now use $w(x)$ as the *weighting function* or *probability distribution* for our random numbers, the integral can be approximated as

$$I = \left\langle \frac{f}{w} \right\rangle \simeq \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{w(x_i)} \tag{11.21}$$

The improvement with (11.21) is that a judicious choice of the weighting function $w(x)$ makes $f(x)/w(x)$ a rather constant function and consequently easier to integrate.

11.7.3 Implementation: Nonuniform Randomness ⊙

In order for $w(x)$ to be the weighting function for random numbers over $[a, b]$, we want it with the properties

$$\int_a^b dx\, w(x) = 1 \qquad (w(x) > 0) \qquad d\mathcal{P}(x \to x + dx) = w(x)dx \tag{11.22}$$

where $d\mathcal{P}$ is the probability of obtaining an x in the range $x \to x + dx$. For the uniform distribution over $[a, b]$, $w(x) = 1/(b - a)$.

- **Inverse Transform/Change of Variable Method**: Let us consider a change of variables that take our original integral I (11.20) to the form

$$I = \int_a^b dx\, f(x) = \int_0^1 dW\, \frac{f[x(W)]}{w[x(W)]} \tag{11.23}$$

Our aim is to make this transformation such that there are equal contributions from all parts of the range in W, that is, we want to use a uniform sequence of random numbers for W. To determine the new variable, we start with $u(r)$, the uniform distribution over $[0, 1]$,

$$u(r) = \begin{cases} 1 & \text{for} \quad 0 \le r \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{11.24}$$

We want to find a mapping $r \leftrightarrow x$ or probability function $w(x)$ for which probability is conserved:

$$w(x)dx = u(r)dr \qquad \Rightarrow \qquad w(x) = \left|\frac{dr}{dx}\right| u(r) \tag{11.25}$$

This means that even though x and r are related by some (possibly) complicated mapping, x is also random with the probability of x lying in $x \to x + dx$ equal to that of r lying in $r \to r + dr$.

To find the mapping between x and r (the tricky part), we change variables to $W(x)$ defined by the integral

$$W(x) = \int_{-\infty}^{x} dx'\, w(x') \tag{11.26}$$

We recognize $W(x)$ as the (incomplete) integral of the probability density $u(r)$ up to some point x. It is in this way another type of distribution function specifically, the integrated probability of finding a random number less than the value x. The function $W(x)$ is on that account called a *cumulative distribution function* and can also be thought of as the area to the left of $r = x$ on the plot of $u(r)$ versus r. It follows immediately from the definition (11.26) that $W(x)$ has the properties

$$W(-\infty) = 0 \qquad W(\infty) = 1 \tag{11.27}$$

$$\frac{dW(x)}{dx} = w(x) \qquad dW(x) = w(x)dx = u(r)dr \tag{11.28}$$

Consequently, $W_i = \{r_i\}$ is a uniform sequence of random numbers and we invert (11.26) to obtain x values distributed with probability $w(x)$.

The crux of this technique is being able to invert (11.26) to obtain $x = W^{-1}(r)$. Let us look at some analytic examples to get a feel for these steps (numerical inversion is possible and frequent in realistic cases).

- **Uniform Weight Function** w: We start off with our old friend, the uniform distribution:

$$w(x) = \begin{cases} \frac{1}{b-a} & \text{if } a \le x \le b \\ 0 & \text{otherwise} \end{cases} \tag{11.29}$$

By following the rules this then leads to

$$W(x) = \int_a^x dx' \, \frac{1}{b-a} = \frac{x-a}{b-a} \tag{11.30}$$

$$\Rightarrow \quad x = a + (b-a)W \qquad \Rightarrow \qquad W^{-1}(r) = a + (b-a)r \tag{11.31}$$

where $W(x)$ is always taken as uniform. In this way we generate uniform random $r : [0,1]$ and uniform random $x = a + (b-a)r : [a,b]$.

- **Exponential Weight:** We want to generate points with the exponential distribution:

$$w(x) \;=\; \begin{cases} \frac{1}{\lambda} e^{-x/\lambda} & \text{for } x > 0 \\ 0, & \text{for } x < 0 \end{cases} \qquad W(x) = \int_0^x dx' \, \frac{1}{\lambda} e^{-x'/\lambda} = 1 - e^{-x/\lambda}$$

$$\Rightarrow \quad x = -\lambda \ln(1-W) \equiv -\lambda \ln(1-r) \tag{11.32}$$

In this way we generate uniform random $r : [0,1]$ and obtain $x = -\lambda \ln(1-r)$ distributed with an exponential probability distribution for $x > 0$.

Notice that our prescription (11.20)–(11.21) tells us to use $w(x) = e^{-x/\lambda}/\lambda$ to remove the exponential-like behavior out of an integrand and place it into the weights and scaled points ($0 \le x_i \le \infty$). Because the resulting integrand will vary less, it may be approximated better as a polynomial:

$$\int_0^\infty dx \, e^{-x/\lambda} f(x) \;\simeq\; \frac{\lambda}{N} \sum_{i=1}^{N} f(x_i) \qquad x_i = -\lambda \ln(1-r_i) \tag{11.33}$$

- **Gaussian (Normal) Distribution**:
  We want to generate points with a normal distribution:

$$w(x') = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x'-\overline{x})^2/2\sigma^2} \tag{11.34}$$

This by itself is rather hard, but it is made easier by generating uniform distributions in angles and then using trigonometric relations to convert these to a Gaussian distribution. But before doing that, we keep things simple by realizing that we can obtain (11.34) with mean $\overline{x}$ and standard deviation σ, by scaling and a translation of a simpler $w(x)$:

$$w(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \qquad x' = \sigma x + \overline{x} \tag{11.35}$$

We start by generalizing the statement of probability conservation for two different distributions (11.25) to two dimensions [9]:

$$p(x,y)dxdy = u(r_1,r_2)dr_1dr_2 \quad \Rightarrow \quad p(x,y) = u(r_1,r_2)\left|\frac{\partial(r_1,r_2)}{\partial(x,y)}\right| \tag{11.36}$$

We recognize the term in vertical bars as the Jacobian determinant:

$$J = \left|\frac{\partial(r_1,r_2)}{\partial(x,y)}\right| \overset{\text{def}}{=} \frac{\partial r_1}{\partial x}\frac{\partial r_2}{\partial y} - \frac{\partial r_2}{\partial x}\frac{\partial r_1}{\partial y} \tag{11.37}$$

To specialize to a Gaussian distribution, we consider $2\pi r$ as angles obtained from a uniform random distribution r, and x and y as Cartesian coordinates that will have a Gaussian distribution. The two are related by

$$x = \sqrt{-2\ln r_1}\cos 2\pi r_2 \qquad y = \sqrt{-2\ln r_1}\sin 2\pi r_2 \tag{11.38}$$

The inversion of this mapping produces the Gaussian distribution

$$r_1 = e^{-(x^2+y^2)/2} \qquad r_2 = \frac{1}{2\pi}\tan^{-1}\frac{y}{x} \qquad J = -\frac{e^{-(x^2+y^2)/2}}{2\pi} \tag{11.39}$$

The solution to our problem is at hand. We use (11.38) with r_1 and r_2 uniform random distributions, and x and y will then be Gaussian random distributions centered around $x = 0$.

- **Alternate Gaussian Distribution**: The central limit theorem can be used to deduce a Gaussian distribution via a simple summation. The theorem states, under rather general conditions, that if $\{r_i\}$ is a sequence of mutually independent random numbers, then the sum

$$x_N = \sum_{i=1}^{N} r_i \tag{11.40}$$

is distributed normally. This means that the generated x values have the distribution

$$P_N(x) = \frac{\exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]}{\sqrt{2\pi\sigma^2}} \qquad \mu = N\langle r\rangle \quad \sigma^2 = N(\langle r^2\rangle - \langle r\rangle^2) \tag{11.41}$$

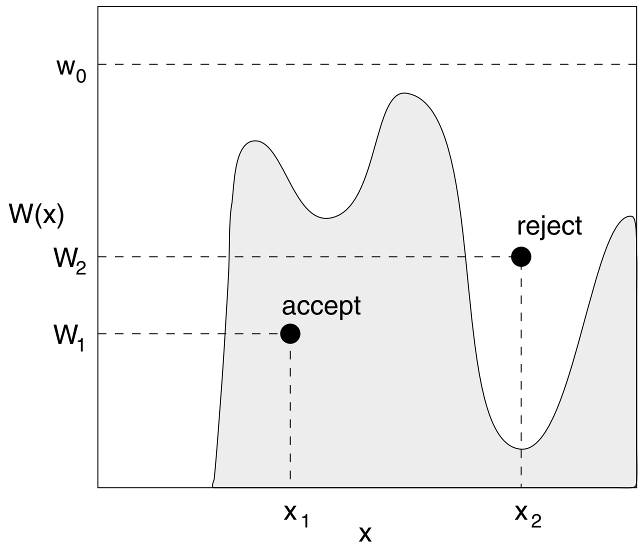

Fig. 11.6 The von Neumann rejection technique for generating random points with weight $w(x)$.

11.7.4 von Neumann Rejection (Method) ⊙

A simple and ingenious method for generating random points with a probability distribution $w(x)$ was deduced by von Neumann. This method is essentially the same as the rejection or sampling method used to guess the area of the pond; only now the pond is replaced by the weighting function $w(x)$ and the arbitrary box around the lake by the arbitrary constant W_0. Imagine a graph of $w(x)$ versus x (Fig. 11.6). Walk off your box by placing the line $W = W_0$ on the graph, with the only condition being $W_0 \geq w(x)$. We next "throw stones" at this graph and count only those that fall into the $w(x)$ pond. That is, we generate uniform distributions in x and $y \equiv W$ with the maximum y value equal to the width of the box W_0:

$$(x_i, W_i) = (r_{2i-1}, W_0 r_{2i}) \tag{11.42}$$

We then reject all x_i that do not fall into the pond:

$$\text{if } W_i < w(x_i) \text{ accept} \qquad\qquad \text{if } W_i > w(x_i) \text{ reject} \tag{11.43}$$

The x_i values so accepted will have the weighting $w(x)$ (Fig. 11.6). The largest acceptance occurs where $w(x)$ is large, in this case for midrange x.

In Unit 12, *Thermodynamic Simulations: The Ising Model,* we apply a variation of the rejection technique known as the *Metropolis algorithm*. That algorithm has now become the cornerstone of computation thermodynamics.

11.7.5 Nonuniform Assessment ⊙

Use the von Neumann rejection technique to generate a normal distribution of standard deviation 1, and compare to the preceding *Box–Muller* method.