$x_0 = -\frac{b}{2c}$ and is

$$g(x_0) = a - \frac{b^2}{4c}.$$

Writing:

$$g(x) = c(x - x_0)^2 + 2\left(a - \frac{b^2}{4c}\right)\frac{1}{2},$$

we have expressed the polynomial as the sum of two positive definite functions with the second in the form of a probability times a *pdf* [$\frac{1}{2}$ is the probability distribution function for a uniform distribution $-1 \leq x \leq 1$]. Thus, after choosing a first random number U_1,

if $U_1 < 2(a - \frac{b^2}{4c})$ select from a uniform distribution $-1 \leq x \leq 1$,

else sample a quadratic random variable, z^2, $z \equiv x - x_0$.

Using

$$\mathcal{N} \equiv \int_{-1}^{1}(x - x_0)^2 dx = \int_{-1-x_0}^{1-x_0} z^2 dz = \frac{2}{3}(1 + 3x_0^2) \tag{2.54}$$

with

$$F(z) = \frac{1}{\mathcal{N}}\int_{-1-x_0}^{z} z^2 dz = \frac{1}{3\mathcal{N}}\left[z^3 + (1 + x_0)^3\right] \tag{2.55}$$

we find

$$x = x_0 + \left[2(1 + 3x_0^2)U_2 - (1 + x_0)^3\right]^{\frac{1}{3}} \tag{2.56}$$

where U_2 is a second uniformly distributed random number. Note that some care is necessary in extracting the cube root since the computer will attempt to take the logarithm of a quantity which may be negative. To avoid this problem, write

$$A^{\frac{1}{3}} = \text{sign}(A)|A|^{\frac{1}{3}}. \tag{2.57}$$

2.4 The Metropolis Algorithm

While the techniques just studied for sampling have a certain elegance, they have one serious flaw: it is only with the greatest of difficulty that they can sample multi-dimensional functions. On the other hand, the Metropolis algorithm is designed with exactly that in mind. Without it, Monte Carlo methods would be much less powerful than they are. It turns out that it also leads to a new way of representing a function which, in turn, leads to a new way of solving physical problems, one very well suited to many-body systems. Those methods will be studied in Chapters 6 and 11.

2.4.1 The Method Itself

The algorithm is based on the concept of a random walk. In the two-dimensional classical random walk, a point is started at the origin of an $x - y$ coordinate system and during successive time intervals is allowed to move one unit in any (rectangular) direction with equal probability. Thus it might move one step to the right, another step to the right, one step up, one down, etc. The probabilistic description of this process is that of an expanding Gaussian probability distribution which, in the limit of some very long time, would cover the entire 2-dimensional space.

If a similar random walker is started off at some point in a multi-dimensional space (a walker is specified by giving values of all of the coordinates in the space) and allowed to take a vectorial step (i.e. all components are changed at once) he will eventually find his way into all parts of the space. However, for the present application, we do not want the walker to simply cover all parts of the space equally, we would like him to spend more time in those regions where the function to be sampled is large. In this way the ensemble of the positions of the walker provide a *realization* of the *pdf* since the probability of finding a walker in a given region should be the same as $g(x_1, x_2, \ldots, x_N)$. In other words, if one could bin up all of the positions of all of the steps of the walker the probabilities would follow the *pdf*.

$$\begin{aligned}
x_1' &= x_1 + \delta\{-1, 1\} \\
y_1' &= y_1 + \delta\{-1, 1\} \\
z_1' &= z_1 + \delta\{-1, 1\} \\
x_2' &= x_2 + \delta\{-1, 1\} \\
y_2' &= y_2 + \delta\{-1, 1\} \\
z_2' &= z_2 + \delta\{-1, 1\} \\
&\vdots \\
x_M' &= x_M + \delta\{-1, 1\} \\
y_M' &= y_M + \delta\{-1, 1\} \\
z_M' &= z_M + \delta\{-1, 1\}
\end{aligned}$$

Figure 2.3: Movement of Walkers

The problem is then how to modify the rules of the walk so that the this occurs. We first show *how* the algorithm works and then discuss *why* it works.

The algorithm works as follows. Suppose that we wish to represent some multi-dimensional *pdf*, $g(\mathbf{R})$. For a walker, say at some point $\mathbf{R}$, we propose a random step (of a uniform limit, δ, in each coordinate) to some point $\mathbf{R}'$. For example, if $\mathbf{R}$ is a 3-dimensional variable for M particles then the movements are outlined in Figure 2.3. In this Figure {-1,1} denotes a random number in the range [-1,1]. Each occurrence of this symbol in these equations indicates a *different* random number.

After proposing the step in this manner, the value of the desired *pdf* is computed at the new and old positions[§], giving $g(\mathbf{R}')$ and $g(\mathbf{R})$ and the ratio

$$q \equiv \frac{g(\mathbf{R}')}{g(\mathbf{R})}$$

is formed. The step is then accepted with probability q. This means that, if q is greater than one, it is accepted. If q is less than one, a random number uniformly distributed between 0 and 1 is generated and only if q is greater than this number is the step accepted.

The step may be proposed in a number of ways, not just the one mentioned above. For example, in a finite interval problem the entire range might be used. In practical cases several walkers are carried along simultaneously in order to cover the space and so that an estimate of the value of the quantity being calculated can be made at each step.

As a simple example let us sample the *pdf*:

$$g(x) = \frac{\pi}{2}\sin \pi x \qquad 0 \le x \le 1.$$

We begin by initializing a set of walkers. In a real problem we would try to start them off in a distribution as closely resembling the expected final distribution as possible, but in this case we will simply take them to be uniformly distributed. We then need to initialize the values of the *pdf* at the "old" points to provide the denominator of q at the time of the first test. The code (without including binning for verification) might look something like:

```
      DIMENSION X(5000),G(5000)      ! room for 5000 walkers
      NWALK=5000                     ! number of walkers
      DELTA=.1                       ! maximum step size
      PI=ACOS(-1.)
      DO 1 I=1,NWALK                 ! initialize loop
      X(I)=RAN3(D)
    1 G(I)=SIN(PI*X(I))              ! normalization doesn't matter
      DO 2 K=1,100                   ! do 100 steps
```

[§]In practice the value of $g(\mathbf{R})$ will be available from the previous step


```
      DO 3 I=1,NWALK                 ! for each walker
      XT=X(I)+DELTA*(2.*RAN3(D)-1.)  ! propose new value
      IF (XT.GT.1.) GOTO 3           ! proposed step
      IF (XT.LT.0.) GOTO 3           ! must be in range
      GT=SIN(PI*XT)                  ! calculate proposed pdf
      Q=GT/G(I)
      IF (Q.LE.RAN3(D)) GOTO 3       ! accept or reject
      X(I)=XT                        ! if accept, update values
      G(I)=GT
    3 CONTINUE
    2 CONTINUE
      END
```

If we add a binning procedure to verify the behavior of the algorithm we have:

```
      DIMENSION X(5000),G(5000),IBIN(21),NTHEORY(21)
      NWALK=5000                 ! number of walkers
      DELTA=.1                   ! maximum step size
      PI=ACOS(-1.)
      DO 1 I=1,NWALK             ! initialize loop
      X(I)=RAN3(D)
    1 G(I)=SIN(PI*X(I))          ! normalization doesn't matter
      DO 4 J=1,20                ! bin starting values
    4 IBIN(J)=0
      DO 5 I=1,NWALK
      J=20.*X(I)+1.
    5 IBIN(J)=IBIN(J)+1
      DO 6 J=1,20
      XMIN=(J-1)*.05
      XMAX=J*.05
    6 NTHEORY(J)=NWALK*(COS(PI*XMIN)-COS(PI*XMAX))/2.
      PRINT 7,(J,J=1,20)
      PRINT 7,(IBIN(J),J=1,20)
      PRINT 7,(NTHEORY(J),J=1,20)  ! print binned results
    7 FORMAT(I3,19I4)
      DO 2 K=1,100                 ! do 100 steps
      DO 3 I=1,NWALK               ! for each walker
      XT=X(I)+DELTA*(2.*RAN3(D)-1.)  ! propose new value
      IF (XT.GT.1.) GOTO 3         ! proposed step
      IF (XT.LT.0.) GOTO 3         ! must be in range
      GT=SIN(PI*XT)                ! calculate proposed pdf
```

```
      Q=GT/G(I)
      IF (Q.LE.RAN3(D)) GOTO 3      ! accept or reject
      X(I)=XT                       ! if accept, update values
      G(I)=GT
    3 CONTINUE
      IF (MOD(K,20).NE.0) GOTO 2    ! bin every 20 steps
      DO 14 J=1,20                   ! start binning
   14 IBIN(J)=0
      DO 15 I=1,NWALK
      J=20.*X(I)+1.
   15 IBIN(J)=IBIN(J)+1
      PRINT 7,(J,J=1,20)
      PRINT 7,(IBIN(J),J=1,20)
      PRINT 7,(NTHEORY(J),J=1,20)   ! print binned results
    2 CONTINUE
      END
```

2.4.2 Why It Works

To understand why the Metropolis algorithm works, think of the flow of probability like the flow of something material. There are some areas more dense than others with a constant flux of material between the areas. There are many possible scenarios for creating flow such that the equilibrium distribution is the one that represents $g(\mathbf{R})$. Our problem is easy, in a sense, because we only need to find one method to control the flow of probability in this dynamic equilibrium such that we reach a given steady state. Actually that is not quite true since, among all of the possible choices, we would like to find the one which would reach the equilibrium limit the most quickly.

Consider the conditions which must prevail when the equilibrium state is reached. If $P(\mathbf{R}',\mathbf{R})$ is the probability of a flow from $\mathbf{R}$ to $\mathbf{R}'$ and $g(\mathbf{R})$ the *pdf* at $\mathbf{R}$ then, since the large probability of flow from the low density to high density must match the small probability of flow from high density to low density (so that the densities remain constant), we must have:

$$P(\mathbf{R}',\mathbf{R})g(\mathbf{R}) = P(\mathbf{R},\mathbf{R}')g(\mathbf{R}') \tag{2.58}$$

This equation simply states that the rate of flow toward high density over the rate of flow away from high density is the inverse of the ratio of the densities.

We can now break the probability $P(\mathbf{R}',\mathbf{R})$ into the product of two factors corresponding to a conditional probability. We shall express this probability by first proposing the change $\mathbf{R} \rightarrow \mathbf{R}'$ and then making a test to see if the proposed change

is accepted. Thus we will rewrite

$$P(\mathbf{R}',\mathbf{R}) = A(\mathbf{R}',\mathbf{R})T(\mathbf{R}',\mathbf{R}) \tag{2.59}$$

where $T(\mathbf{R}',\mathbf{R})$ is the probability that we propose a point near $\mathbf{R}'$ starting at $\mathbf{R}$ and $A(\mathbf{R}',\mathbf{R})$ is the probability that such a "move" will be accepted. Any combination "AT" which satisfies Eq. 2.58 will work, some may work faster than others, some may be easier than others to implement. To simplify our task we choose

$$T(\mathbf{R}',\mathbf{R}) = T(\mathbf{R},\mathbf{R}'). \tag{2.60}$$

The balance equation 2.58 is now reduced to:

$$A(\mathbf{R}',\mathbf{R})g(\mathbf{R}) = A(\mathbf{R},\mathbf{R}')g(\mathbf{R}'). \tag{2.61}$$

It only remains for us to pick a function "A" which satisfies this equation. It is useful to define the quantity,

$$q(\mathbf{R}',\mathbf{R}) = \frac{g(\mathbf{R}')}{g(\mathbf{R})}. \tag{2.62}$$

Consider the function:

$$A(\mathbf{R}',\mathbf{R}) \equiv \min[1, q(\mathbf{R}',\mathbf{R})]. \tag{2.63}$$

By examining Eq. 2.61 for the case that $q < 1$ and $q > 1$ it can be seen that is satisfied in both situations. It is also seen that the condition for acceptance is indeed the one outlined in the previous section.

There is an alternate form:

$$A(\mathbf{R}',\mathbf{R}) \equiv \frac{q(\mathbf{R}',\mathbf{R})}{1+q(\mathbf{R}',\mathbf{R})} \tag{2.64}$$

which also works.

2.4.3 Comments on the Algorithm

One thing to note is that the normalization of the *pdf* never enters into the equations. This is convenient in that it is not necessary to know specifically how to normalize the functions. It is very useful if one is dealing with a multidimensional probability densities having a complicated form. In fact, it is exactly in this instance that the method finds its greatest utility. However, if we wish to compute an integral which is not really in the form of Eq. 2.2, then we must know the normalization by other means. For example, if we wished to do the integral in Eq. 2.2 by sampling the entire integrand, as we did in Section 2.3.9, we would simply obtain unity. Of course that is what we obtained in that instance for the sum, but in that case the sampling method

was capable of returning the normalization of the function at the same time since it was a rejection technique. This feature of the Metropolis algorithm is a great benefit or a difficulty depending on the type problem to be solved.

The vector containing the walkers represents the *pdf* in the sense that more walkers are concentrated in the region where the function is large. This can be a very useful way to represent a function. For simple functions it may seem like a poor (and far-fetched) expression but for multidimensional functions it is very efficient. If we wish to average over a function, the values are only needed where it is large. The usual representation *at all points in the space* becomes incredibly wasteful. Consider a mesh of only 10 points in each dimension. For thirty particles in three dimensions we would need 10^{90} tabulated values. This number far exceeds any memory one could expect to have (indeed it far exceeds the number of particles in the universe). With the Monte Carlo representation, problems with the order of a thousand particles can be solved. The direct utility of this representation of functions is limited by the fact that they can be used only to calculate integrals. There are, however, special techniques which can be employed to calculate the desired quantities with these functions. These techniques are treated in Chapter 11.

Problems

Problems marked with * are meant to be solved without the aid of a computer.

1. Code the Monte Carlo solution for the area of a circle to find π. Extend it to 3 dimensions to get π from the volume of a sphere $\frac{4\pi}{3}$. If you use `RAN3` call it several times between usages for the second part of this problem to avoid the correlations among 3 successive values.

2. Integrate x, x^2, x^3, x^4 and x^5 from 0 to 1 using Monte Carlo with x chosen uniformly from 0 to 1.

3. Generate distributions for $h_n(x) = (n+1)x^n \quad 0 \le x \le 1$, put them in bins and compare with the expected *pdf* for $n = 1,\ 2$ and 3.

4. Perform the integral:

$$\int_0^1 x^5 dx = \frac{1}{n+1}\int_0^1 h_n(x)x^{5-n}dx$$

for $n = 0,\ 1,\ 2,\ 4,\ 5$ by Monte Carlo. Compare the relative accuracy by examining the variance computed from Eq. 2.7

5. * Write an equation to sample x from the normalized probability distribution function

$$g(x) = \frac{3x^2}{(1+x^3)^2} \qquad 0 \le x \le \infty.$$

Note that

$$\int_0^x g(x)dx = \frac{x^3}{1+x^3}.$$

6. * Derive the normalized form, the cumulative distribution function, $F(x)$, and the sampling equation for:

$$g_3: \quad \int \frac{x\,dx}{(a^2+x^2)^{n+1}} = -\frac{1}{2n(a^2+x^2)^n}$$

and

$$g_4: \quad \int \frac{dx}{(a^2+b^2x)x^{\frac{1}{2}}} = \frac{2}{ab}\tan^{-1}\frac{bx^{\frac{1}{2}}}{a}.$$