# Monte Carlo Calculation

Monte Carlo techniques are very useful in solving a variety of computational problems. We shall begin with a discussion on generating random numbers, the starting point of any Monte Carlo calculation.

## 27 Generation of random numbers

### 27.1 Uniformly distributed random integers

In Monte Carlo calculations, we usually have to take random samples of certain quantities with definite distributions. For this reason, the random

number we use should follow the same distribution. For example, in a one-dimensional Monte Carlo integration, we wish to sample the variable of integration with uniform probability within the integration limits. For this purpose, we need random numbers with a uniform distribution.

A simple way to obtain a random integer is to read the timer, or system clock, on the computer we are using. Let us assume, for the convenience of discussion here, that the system clock is measured in units of microsecond ($\mu s$) and is set to zero at some arbitrary time. A simple function can be written to extract the time as integers in $\mu s$ very time it is called. If $t_1$ and $t_2$ are the values obtained in two successive calls separated be a few seconds, the most significant digits of the two integers will be the same, as they represent the time in units much larger than a second. On the other hands, the three least significant digits, for example, are likely to be random. The source of the randomness in this case comes from the

fact that it is not possible for us to control our reaction time to be more accurate than the order of 0.1s. If we obtain a sequence of such values, $t_1, t_2, \cdots$, than

$$r_i = t_i \bmod 1,000$$

for $i = 1, 2, \cdots$ is a sequence of random integers in the range of $[0, 999]$.

Another interesting way to produce random integers is the middle-square method of von Neumann. If we square an integer consisting of several digits, both the most significant digits and the least significant digits are predictable. However, it is more difficult to predict the digits in the middle. For example, if we square an integer of three digits,

$$(123)^2 = 15,129$$

the result is a five-digit integer. By chopping off the leading and trailing digit, we obtain the number 512. If we square this number and retain only the third, fourth, and fifth digits, we obtain the number 214. In this way, a sequence of random integers can be obtained, each constructed from the square of the previous one with only the middle part of the digits retained. Although this method has been in use for a long time, it is no longer a preferred way to generate random integers. Tests have shown that there are many ways such a sequence can develop into a bad direction. For example, if for any reason a member of the sequence becomes zero, all the remaining members of the sequence will obviously be zero.

## 27.2 The linear congruence method

The most popular way to generate random integers with a uniform distribution is the linear congruence method. In this approach, a random integer $X_{n+1}$ is produced from another one $X_n$ through the operation

$$X_{n+1} = (aX_n + c) \bmod m \tag{1}$$

where the modulus, $m$, is a positive integer. The multiplier $a$ and increment $c$ are also positive integers but their values must be less than $m$. To start off the sequence of random integers $X_0$, $X_1$, $X_2$, .." we need to input an integer $X_0$, generally referred to as the seed of the random sequence,

The spirit of Eq. 1 is very similar to that behind the system-clock and middle-square methods discussed above. Instead of taking the square, the

seed is multiplied by an integer $a$, and to prevent a bad sequence from developing, an increment $c$ is added to the product. Similar to the system-clock approach, only the least significant part of the sum (less than $m$) is retained through the modulus operation.

The quality of random integers generated by the linear congruence method depends critically on the values of $m$, $a$, and $c$ selected. Because of the modulus operation, all the random integers produced by this method must be in the range $[0, m-1]$. For this reason, we want $m$ to be as large as possible. On the other hand, $m^2$ cannot be larger than the longest integer the computer can store in its memory. It is also clear that the choices of these three quantities are related with each other, For example, in order to have a long period, it is known that

- $c$ must be relative prime to $m$ ($a$ and $m$ share no common factors other than 1).

- $(a-1)$ is a multiple of $p$, if $p$ is a prime factor of $m$, and

- $(a-1)$ is a multiple of 4, if $m$ is a multiple of 4.

For example, a possible set of values for computers with word length of 32 bits is

$$m = 714,025 \quad a = 1,366 \quad c = 150,889.$$

The maximum number of different random numbers that can be generated is 714,205 in this case.

## 27.3 Conversion to random numbers

If instead of random integers we wish to have random (floating) numbers, we can convert an integer $X_n$ in the interval $[0, m]$ to a floating number $Rn$ in interval $[a, b]$. For most generators, the interval is taken to be $[0, 1]$. If the largest integer produced by the linear congruence method of Eq.1 is $m$, any random integer $X_n$ may be converted to a random number in the interval $[0, 1]$ by dividing $X_n$ with $m$. To carry out this procedure on a computer, we must first change both $X_n$ and $m$ into floating numbers and then take the quotient

$$R_n = X_n/m.$$

Note that, since $X_n$ in Eq. 1 is needed to generate the next random integer, it must be preserved in a computer code. Because of the wide range of

applications of random numbers, computer operating systems and high-level languages are usually equipped with random number generators. A slight variant of Eq. 1 is often found on 32-bit machines,

$$X_{n+1} = \frac{aX_n + c}{d} \bmod m \tag{2}$$

with

$$\begin{aligned} m &= 32,768 \quad a = 1,103,515,245 \\ c &= 12,345 \quad d = 65,536. \end{aligned}$$

The range of random integers generated by this method is $0 < X_n < 2^{15}$. There is, however, a small problem if we wish to implement Eq. 2 using a language such as Fortran. Since the integer $a$ is larger than $2^{30}$, there is a high probability that an overflow will take place on a 32-bit word length

computer when it is multiplied by a random integer $X_n$. To avoid the problem this portion of code can be written in a different language such as C in a way that it may be called by a Fortran program.

## 27.4 Improving randomness by shuffling

One weak point of the linear congruence method is that every random integer is generated from the one before it. As a result, there is a strong possibility for the random numbers to be sequentially correlated. One consequence of such a correlation may be illustrated by the following example. If we use groups of three random numbers ($R_{3n}$, $R_{3n+1}$, $R_{3n+2}$) as the coordinates $(x, y, z)$ of points inside a cube of length) in arbitrary units

on each side, there is the tendency for the points to fall mainly on a small number of surfaces inside the cube rather than filling up the volume evenly.

In spite of such a shortcoming, the linear congruence method is widely used, as it is an efficient way to generate random numbers. Several remedies have been introduced to overcome its weakness. A simple method to alleviate the correlation tendency is to shuffle the sequence. That is, a large number of random numbers is generated by the linear congruence method and stored in an array. Let the size of the array be $L$, preferably a prime number. Anytime we need a random number, we take it from the array rather than having it generated directly. However, instead of taking them in sequence, a random integer $j$ in the range $[1, L]$ is generated. The required random number is then taken from the $j^{th}$ location of the array. To prevent the same random number from being used again, the $j^{th}$ location in the array is replaced by a new random number produced

**Box 7-1 Function** RSHFL(ISEED)
**Improved random number generator by shuffling**

Argument list:
ISEED: Random number seed on input. Returns the seed for the next one.
Initialization:
(a) Select a stack $S(j)$ of length $L = 97$.
(b) Store $S(j)$ with random numbers produced by a linear congruence generator.
(c) Mark the stack as initialized.

1. Make sure that the stack is initialized.
2. Generate a new random number $X$.
3. Convert $X$ to an integer $j$ in the range $[1, L]$.
4. Return $S(j)$ as the random number required.
5. Replace $S(j)$ by $X$.

Figure 1:

by the generator. This is implemented in the algorithm of Box 7-1. For simplicity, we can use the linear congruence generator to fill the array, as the most serious problem of correlation is now corrected by the shuffling.

## 27.5 Alternate procedures

Another way to improve the linear congruence method is to construct a new formula that makes use of more than one previous member of the sequence. Instead of $X_n$ alone, as in Eq. [**?**], we can include an additional member on the right side of the equation,

$$X_{n+1} = (X_n + X_{n-k}\} \bmod m$$

with $k$ being a number larger than 15 or so to avoid possible correlations between $X_n$ and $X_{n-k}$. For example,

$$X_{n+1} = (X_{n-24} + X_{n-55}) \bmod m \qquad \text{for } n > 55.$$

This is known as an additive generator. A variant of this is often used in practice is the subtractive generator ,

$$X_{n+1} = (X_{n-55} - X_{n-24}) \bmod m.$$

Among other advantages, this method can be made completely portable. That is, the computer program may be written in a high-level language, such as Fortran or C, and can be made to run on any machine that has a compiler for the language.

## 27.6 Random numbers with a given distribution

In many calculations, we need random numbers with a distribution that is different from a uniform one. The general approach is to start with

a uniform set and change it into the one required. Assume a uniform probability distribution in the range of $[0, 1]$,

$$f(x)dx = \begin{cases} dx & 0 < x < 1 \\ 0 & \text{otherwise} \end{cases}$$

The probability is of course normalized, so that

$$\int_{-\infty}^{+\infty} f(x)dx = 1.$$

Now suppose that we generate a uniform deviate $x$ and then take some prescribed function of it $y(x)$. The probability distribution of $y$, denoted $p(y)dy$, is determined by the fundamental transformation law of probability, which is simply

$$|p(y)dy| = |f(x)dx|$$

or

$$p(y) = f(x) \left| \frac{dx}{dy} \right|$$

As an example, suppose that $y(x) = -\ln x$. Then,

$$\begin{aligned} p(y)dy &= f(x)dx \\ &= f(x) \left| \frac{dx}{dy} \right| dy \\ &= e^{-y} dy \end{aligned}$$

which is distributed exponentially. This exponential distribution occurs frequently in real problem, such as the radioactive decay of nuclei.

The transformation methods generalize to more than one dimension. If the $x_1, x_2, \cdots, x_n$ are random deviates with a joint distribution

$$f(x_1, x_2, \cdots, x_n) dx_1 dx_2 \cdots dx_n$$

and if $y_1, y_2, \cdots, y_n$ are each function of all $x$'s, then the joint probability distribution of $y$'s is

$$\begin{aligned} & p(y_1, y_2, \cdots, y_n) dy_1 dy_2, \cdots dy_n \\ = \quad & f(x_1, x_2, \cdots, x_n) \left| \frac{\partial(x_1, x_2, \cdots, x_n)}{\partial(y_1, y_2, \cdots, y_n)} \right| \\ & \times dy_1 dy_2, \cdots dy_n \end{aligned}$$

where $\left| \frac{\partial(x_1, x_2, \cdots, x_n)}{\partial(y_1, y_2, \cdots, y_n)} \right|$ is the Jaccobi determinant of $x$'s with respect to $y$'s.

## 27.7 Random number with a normal distribution

An important example is the normal distribution,

$$p(y)dy = \frac{1}{(2\pi)^{1/2}} e^{-y^2/2} dy.$$

There is no a simple transformation to connect it to the uniform distribution.

Let us consider a transformation between $x_1, x_2$ and $y_1, y_2$ with

$$\begin{aligned} y_1 &= (-2 \ln x_1)^{1/2} \cos(2\pi x_2) \\ y_2 &= (-2 \ln x_1)^{1/2} \sin(2\pi x_2) \end{aligned}$$

Or equivalently

$$\begin{aligned} x_1 &= \exp\left[-\frac{1}{2}(y_1^2 + y_2^2)\right] \\ x_2 &= \frac{1}{2\pi}\arctan\frac{y_2}{y_1} \end{aligned}$$

The Jaccobi determinant can be calculated

$$\begin{aligned} \frac{\partial(x_1, x_2)}{\partial(y_1, y_2)} &= \begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \frac{\partial x_1}{\partial y_2} \\ \frac{\partial x_2}{\partial y_1} & \frac{\partial x_2}{\partial y_2} \end{vmatrix} \\ &= -\frac{1}{(2\pi)^{1/2}} e^{-y_1^2/2} \frac{1}{(2\pi)^{1/2}} e^{-y_2^2/2} \end{aligned}$$

Since this is the product of a function of $y_1$ alone and a function of $y_2$ alone, we see that each $y$ is independently distributed according to the normal distribution.

## 27.8 Rejection method

If the desired distribution of random numbers is not given by an analytical function, or if the inverse function for the indefinite integral of $P(x)$ does not exist, the transformation method fails. In such cases, the rejection method may be used. The basic idea here is quite simple. If we throw away those that are outside $P(x)$, the remainder must be distributed in the way we want. This type of approach was used, in part, in generating random numbers with a normal distribution above. Consider a two-dimensional surface. As we have seen earlier, the location of each point on this surface may be specified by the values of its x- and y-coordinates with respect to some fixed system of reference. A random point on this surface may be chosen by selecting two random numbers $X_1$ and $X_2$, and let the coordinates $x = X_1$ and $y = X_2$. If both $X_1$ and $X_2$ are uniformly

distributed in the interval $[0, 1]$, the random point has an equal probability of being anywhere in the square area. If a large number of such random points are available, the square area will be uniformly covered. On the other hand, if we reject those points with $y > P(x)$, where $P(x)$ is the desired distribution, the remaining points fill only the area underneath the curve $P(x)$. Consider now only the distribution of the values of the x-coordinates of the points remaining. Since the ratio of the numbers of points in two small intervals, one at $x = a$ and the other at $x = b$, is given by $P(x = a)/P(x = b)$, the distribution of $x$ values is given by the probability $P(x)$ dx. In more practical terms, if we have a collection of $N$ points, represented by their x- and y-coordinates $\{X_1(i), X_2(i)\}$, for $i = 1, 2, ..., N$, and we discard all those points [both $X_1(i)$ and $X_2(i)$] with $X_2(i) > P(x = X_1(i))$, the remaining random numbers $X_1(i)$ have the desired distribution $P(x)$.

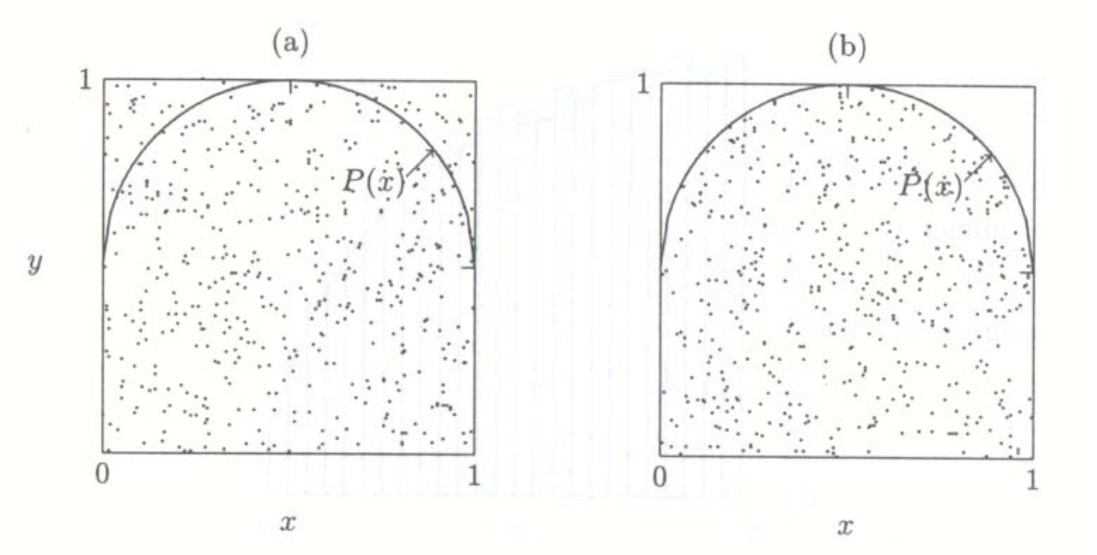


Figure 2: (a) Random points, each specified by two random numbers as its x-axis x- and y-coordinates, uniformly filling a square area. (b) When points with y $>$ P(x) are rejected, the x values of remainder have distribution P(x).

A simple example may be helpful here to illustrate the principle behind the rejection method. Let us consider the semicircular distribution

$$P(x) = \frac{2}{\pi}(1 - x^2)^{1/2}$$

For $x > 0$, the distribution covers only a quarter of a circle of unit radius with the center of the circle located at the origin. To produce random numbers having such a distribution with the rejection method, we start out with a uniform set in interval $[0, 1]$. For each pair of such random numbers, $X_1(i)$ and $X_2(i)$, we shall consider $X_1(i)$ as the x-coordinate and $X_2(i)$ as the y-coordinate of a random point in a square area of unit length on each side. If $X_l^2(i) + X_2^2(i) > 1$, the point is outside the quarter-circle and we reject the point. Another point is produced by taking a new pair of random numbers $X_1(i+1)$ and $X_2(i+1)$. If $X_1^2(i+I)+X_2^2(i+I) < 1$, the point is inside the unit circle and we retain $X_1(i+1)$ as one of the random numbers with a semicircular distribution. The method is essentially the same as that

used earlier to produce random numbers with a circular distribution that serve as the intermediate quantities to generate random numbers with a normal distribution.

Reference

http://farside.ph.utexas.edu/teaching/329/lectures

# 28 Monte Carlo techniques

Earlier we looked at various methods for calculating integrals. All of these methods used some form of polynomial approximation for the integrand.

However, there are other methods to calculate these integrals, many of which do not rely on approximation. One such method is known as the Monte Carlo method, since it uses random numbers.

In general, any computational procedure that relies on the use of random number generation to determine the results of a calculation is classified as a Monte Carlo procedure. However, Monte Carlo techniques have a much broader usefulness than simple integral calculations. Any physical event that can be viewed as a stochastic process is a prime candidate for modeling via Monte Carlo techniques. This includes statistical physics, high-energy particle physics, and experimental modeling. For this reason, Monte Carlo methods play a major role in computational physics and represent a significant addition to the computational physicist's toolbox.

## 28.1 Acceptance-Rejection Integration

In earlier chapters we explored various methods of performing numerical integration. All of these methods depended on expanding the integrand in some form, usually as a power series. There are other ways of calculating the integral however. Many of these are based on Monte Carlo techniques. In particular, Monte Carlo methods are extremely useful in evaluating multidimensional integrals, where an expansion of the integrand can be very difficult.

We have already seen the simplest form of Monte Carlo integration, which is similar to the acceptance-rejection method. Recall that this method was used to generate random numbers that satisfied the probability distribution $P_x(x)$. A set of uniformly distributed random numbers are generated to

create a point $(x_i, y_i)$ where $A \leq x_i < B$ and $0 \leq y_i < P_x^{\max}$. If $y_i < P_x^{\max}$, then $x_i$ is kept as an acceptable random variable. Similarly, if we replace $P_x(x)$ with the integrand $f(x)$ and $P_x^{max}$ with a constant function $C$, where $C$ is usually the maximum (or minimum, if $f(x)$ is negative over the full range) value of $f(x)$ over the range $[A, B)$, then we can define a rectangle whose area is easily calculated. Recall that the value of the integral is simply the area enclosed by the curve and the abscissa. This area can be approximated by calculating the ratio

$$\frac{\int_A^B f(x)dx}{C(A-B)} = \frac{N[y_i(x_i) \leq f(x_i)]}{N_{tot}}$$

where $N[y_i(x_i) \leq f(x_i)]$ is the number of points satisfying $y_i(x_i) \leq f(x_i)$, $N_{tot}$ is the total number of points generated, and $C(B-A)$ is area of the bounding rectangle. Thus, the integral is calculated as

$$\int_A^B f(x)dx = \frac{N[y_i(x_i) \leq f(x_i)]}{N_{tot}} C(A-B) \tag{3}$$

From equation ([?]) it is apparent that the greater the number of points generated, the more accurate the value calculated for the integral.

## 28.2 Mean Value Integration

Another integration method that is similar to the acceptance-rejection approach is the mean value method. In this approach, we again use the fact that the integral is simply the area under the curve. This time however, we want to use the mean value theorem, which stated that the integral of $f(x)$ over the interval $B - A$ is simply equal to the average value of $f(x)$ over the range multiplied by the interval,

$$\int_A^B f(x)dx = (B - A)\langle f \rangle$$

Using the mean value theorem, the problem reduces to one of simply calculating the average value of $f(x)$.

Imagine that we generate a list of uniform deviates distributed between $A$ and $B$. To calculate the average of the function, simply evaluate $f(x)$ at each of the randomly selected points, and divide by the number of points

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

As the number of points increases, the above average tends towards the real average of $f(x)$. Thus, the integration equation becomes

$$\int_A^B f(x)dx = \frac{B-A}{N} \sum_{i=1}^{N} f(x_i) \tag{4}$$

Example:

Calculate the integral $\int_0^{10} dx/x^2$ along with the error (since we know that this evaluates to 0.9).

This can be accomplished via the following code:

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main()
{
 unsigned long initial_seed = 42;
 double x_i, sum = 0.0, error, average;
 long N, i, j;
```

```
/* Initialize the random number generator */

 rand(initial_seed);

/* Calculate the function a total of 100,000 times
 printing an estimate of the integral every 100 times. */

 for (i = 0; i < 1000; i++)
 {

/* The outer loop controls the printing, so include an
 inner loop to control the additional evaluations */

 for (j = 0; j < 100; j++)
 {
```

```
 x_i = rand();
 sum += 1.0 / (x_i * x_i);
 }

/* The function has been evaluated (100 * i) times, so
 print out intermediate result */

 N = i * 100;
 average = (9.0 / (double)N) * sum;

/* Calculate the relative error from the known value of the
 integral */

 error = ((((average -- 0.9) < 0) ? (0.9 -- average) :
 (average -- 0.9)) / 0.9);
```

```
 printf(‘‘Monte Carlo average = f\nerror = f\n’’,
 average, error);
 }
}
```

## 28.3 Importance Sampling

We can improve the accuracy of the Monte Carlo integral if we know something about the integral a prior. In particular, if we can identify those regions of the function that have the greatest impact on the integral, then we can insure that those regions receive the majority of random points.

This is accomplished by identifying a new function, $g(x)$, such that $g(x) \simeq f(x)$, and then instead of integrating $\int_A^B f(x)dx$, integrate

$$\int_A^B \frac{f(x)}{g(x)} g(x)dx = \int_{y^{-1}(A)}^{y^{-1}(B)} \frac{f(x)}{g(x)} dy$$

where $y(x) = \int^x g(t)dt$. Thus, instead of uniformly sampling $x$ to integrate $f(x)$, we uniformly sample $y$ and integrate $f(x)/g(x)$. Since , the integrand will be approximately unity and easy to evaluate. Notice that this is exactly the procedure that we used in finding an invertible distribution to describe non-uniform random number distributions! In this case, $g(t)$ is used in place of the desired distribution and $y(x)$ replaces the cumulative distribution function.

Example: evaluate the integral

$$I = \int_{-\infty}^{+\infty} e^{-x^2/2} dx$$

Make a transformation,

$$\begin{aligned} y &= e^{-x^2/2} \\ x &= \pm \left(\ln(1/2y)\right)^{1/2} \\ dy &= -e^{-x^2/2} x dx \end{aligned}$$

The integral becomes

$$I = 2 \int_0^1 \left(\ln(1/2y)\right)^{-1/2} dy$$

## 28.4 Variance Reduction

Another method to improve the integration relies upon the fact that when the function doesn't differ much from its average then the standard Monte Carlo approach will work without many points. Unfortunately, most functions aren't very flat over the full range of interest. In these cases, many of the random numbers pairs are either rejected, or else fall in regions where they make little contribution to the integral.

We can improve the method by mapping the function $f(x)$ into another function $g(x)$, with the stipulation that $g(x)$ has a smaller variance than $f(x)$ over the interval in question. In particular, suppose that we construct a function $g(x)$ that satisfies

$$|f(x) - g(x)| \leq \varepsilon \tag{5}$$

and

$$\int_A^B g(x)dx = I_0 \tag{6}$$

where $\varepsilon$ is some predetermined upper limit, usually a fraction of the variance associated with $f(x)$, and $I_0$.

If we can find a $g(x)$ which satisfies equations ([**?**]) and ([**?**]), then we can use the Monte Carlo method to evaluate

$$\int_A^B [f(x) - g(x)]\, dx$$

The final result is found by adding $I_0$ to the result,

$$\int_A^B f(x)dx = \int_A^B [f(x) - g(x)]\, dx + I_0$$

As a few simple tests show, Monte Carlo methods for integrating a function are usually not the prime choice. Indeed, if the integral can be done by any

other means, those means should be employed. Monte Carlo methods are the best choice in those situations where the integral is difficult, or even impossible, to evaluate in any other way.

## 28.5 Uncertainties

If the Monte Carlo methods are not recommended, why use them? The answer to this lies in the nature of the uncertainties associated with the results. Recall that every integration method has an uncertainty associated with it, typically some power of the step size. Unless the step size can be reduced, this uncertainty cannot be significantly changed without using a different method. This uncertainty is tied to the truncation of the power

series used to approximate the integrand. In contrast, the uncertainties generated by the Monte Carlo method have their roots in probability theory.

Consider a Monte Carlo estimate of an integral obtained using the first 1000 random numbers generated. If we make another estimate of the integral, using the next 1000 random numbers, we would find that the estimated value of the integral is different from the first value. This result should be obvious – a different set of random numbers should yield in a different estimate, although the two estimates might not be very different.

Extending this idea to more and more estimates, we would expect a smooth distribution of estimates to be observed, with most of them near the true value of the integral and with the number of estimates decreasing as we moved away from that true value. This distribution, which is a Gaussian distribution for an infinite number of samples, follows directly from the

fact that the results of the calculation are based upon randomly distributed numbers, and so each calculation is essentially "independent" of any other calculation.

What would happen if we used more random numbers to estimate the integral? Naturally, we would expect to get a "better" answer, in the sense that if a large number of estimates of the integral were made, the distribution of estimates would be narrower about the true value. We can quantify these results by again appealing to probability theory. The uncertainty associated with the best estimate of the value of the integral, which is directly proportional to the width of the distribution, is also proportional to $1/N^{1/2}$, where $N$ is the number of random numbers used. From this we see that if we quadrupled the number of points, we can half the width of the distribution. More specifically, we can calculate the width of the

distribution by finding the standard deviation of the mean associated with the estimate

$$\sigma_x^2 = \frac{1}{N-1}\left[\frac{1}{N}\sum_{i=1}^{N} f^2(x_i) - \left(\frac{1}{N}\sum_{i=1}^{N} f(x_i)\right)^2\right] \tag{7}$$

Notice that equation ([?]) is simply the standard deviation, $\sigma_x$, divided by the square root of the number of samples. Because of this, equation ([?]) can be used to assign a probabilistic meaning to the uncertainties associated with the Monte Carlo method. Since 68.3 of all estimates lie within one standard deviation of the mean, we can also say that there is a 68.3 probability that our particular estimate $\langle f \rangle_N$ lies within one standard deviation of the exact average $\langle f \rangle$.

It is important to realize that the error is accumulated on the fly, being updated as more points are sampled. That is, the two sums in equation

([?]) are updated with each additional point, and the error can be evaluated whenever it is needed. Implicit in this fact is one of the strengths of the Monte Carlo method, namely that if a more accurate estimate of the integral is desired, we only need to sample the integrand at more randomly selected points. As $N$ increases, $\sigma_x$ decreases, and the probability of the result being within some specified distance of the correct result increases. But equation ([?]) also demonstrates the weakness of the method – the improvement goes only as the square root of N.

The error in a Monte Carlo calculation is fundamentally different from that in the other methods of integration that have been discussed. For example, consider the trapezoid rule. In this case, the error represents the inability of the linear approximation in fitting the actual integrand that is being integrated. By making the step size smaller, the fit is made better and the error decreases.

If we were applying the trapezoid method to a one-dimensional integral, since the error is proportional to $\tau^2$, where $\tau$ is the step size, then the error can be cut in half by decreasing the step size by the square root of 2, which corresponds to an increase in the number of steps by the square root of 2. If the integral were two-dimensional, the step size in both dimensions would have to be increased, so the total number of steps would be increased by a factor of 2. This can be extended to any number of dimensions. Let the number of dimensions associated with the integration be D. Then in order to gain a factor of two in the uncertainty, the number of steps would have to be increased by a factor of $2^{D/2}$.

In contrast, since the error associated with a Monte Carlo calculation is probabilistic, as more points are included, the average gets better, regardless of the number of dimensions. To perform a multidimensional integral, the random number generator will be called D times to get each of the

coordinate values. Then the function is evaluated and the sums updated. Although the method converges slowly, this convergence is dependent only on the probabilistic nature of the method, and not on the dimensionality of the integral. Thus, to reduce the error by 2, N need only be increased by 4, regardless of the number of dimensions. This is comparable to the trapezoid rule in four dimensions, and is better than the trapezoid method if $D > 4$. Such integrals occur in numerous areas of physics, including quantum mechanics, quantum field theory and statistical mechanics, among others.

## 28.6 Simulations

In addition to evaluating integrals, Monte Carlo methods are widely used to simulate stochastic processes. Not surprisingly, most of the applications

of stochastic methods are in statistical mechanics and quantum processes. They are frequently used in statistical mechanics since the very nature of the subject involves averaging over a large ensemble of the system, i.e., averaging over many "copies" of the system, which is then allowed to vary from copy to copy. The law of large numbers insures that these variations are Gaussian in nature, so that they become essentially random. In quantum processes, since only the probability of a result can be determined, random numbers can be used to select between the various possibilities.

Monte Carlo simulations also play an important part in experimental design, particularly in high-energy physics. In this case the physical parameters associated with the apparatus is described in one portion of the program. Another portion describes the various particles that can propagate through the apparatus, along with the interaction and decay properties of these particles. These simulations operate by starting with a single

particle. Unless the experiment can guarantee that the initial particle's energy and momentum is always the same, random numbers are generated to determine the initial energy and momentum. These numbers can either be uniformly distributed or else follow some predetermined distribution. The particle's path is projected over some time step, with its endpoint determined either by the distance calculated from $p\tau/m$ (where $p$ is the particle's momentum, $\tau$ is the time step and $m$ is the particle's mass), the particle's average interaction length, or the distance to the next medium, whichever is shortest. In the each case, at the end of the step the probability of an interaction is looked up, and a random number is generated. If this number is less than the interaction probability then an interaction is assumed to have occurred, and the results of the interaction are determined. Since there is usually more than one possible result associated with an interaction, another random number is generated to determine which specific result occurs. When the interaction results in the creation

of secondary particles, the parameters associated with the parent particle are stored, the first of the daughter particles is selected, and its parameters are loaded. This particle is then stepped through the apparatus, with its position, energy and momentum followed until either it generates daughter particles, it falls below some specific cutoff energy, or it leaves the simulation's master volume. This continues until a daughter particle is generated that either falls below the experiment's detection parameters or that leaves the master volume. At this point, the next daughter particle immediately up the chain is selected and followed, again continuing until all of the particles have become undetectable for whatever reason.

We are usually interested in how the experiment will react to the passage of particles through it. Thus, it is important to specify which portions of the experiment can actively detect particles, and which passively interact with them. The response of the detectors to the particles must be specified in

addition to their physical parameters. These responses are then stored for later use. By running the simulation many times, the detector responses can be averaged, and the parameters relevant to the experiment can be determined.

The nature of this type of Monte Carlo simulation lends itself very naturally to object oriented programming. Each part of the apparatus can be described as an individual object, building up a hierarchy of classes that inherit the more general properties from basic templates. Similarly, each type of particle can be treated as a different class, whose interactions are described by specific methods. The inherent strength of this approach has been recognized by the high energy physics community.

Monte Carlo simulations play an important role in experimental design. Since these experiments tend to be very expensive, it is critical to insure

that the experiment is optimally designed to accomplish the task for which it is intended. Monte Carlo simulations allow various design alternatives to be tested before any hardware is built, guiding the construction and optimizing the detector layout. Since many experiments are now so complicated in nature that analytical techniques cannot adequately describe them, Monte Carlo methods are essential to study the expected results. Critical in this task is verifying that the Monte Carlo simulation is accurate. This necessitates comparison of the output of the actual detectors under controlled conditions to the results predicted by the simulation.

## 28.7 A simple example: molecular diffusion and the Brownian motion

We shall examine the molecular diffusion as a random walk

Model:

- the mean free path $l$ as a step length.
- the time interval $\tau$ for each step
- In a time interval $\Delta t$, the number of steps $n = \Delta t/\tau$
- Initial value condition: $x = 0$ at $t = 0$
- Probabilities to the right or left sides are equal, $1/2$

- At a later time $t = n\tau$, it locates at $x(t) = ml$, m is the position away from $x = 0$.The probability is

$$p(n, m) = \frac{n!}{(\frac{n+m}{2})!(\frac{n-m}{2})!}(\frac{1}{2})^n$$

  Results:

- For a large number of particles, the mean value of $x(t)$ : $\langle x(t) \rangle = 0$

- The variance of the distribution: $var(x(t)) = \left\langle x(t)^2 \right\rangle - \langle x(t) \rangle^2 = ln(= t/\tau)$

  Two results:

$$\langle m \rangle = 0$$

$$\left\langle m^2 \right\rangle = n$$

## 28.8 Percolation

Percolation

Simulation of square lattice

# 29 The Metropolis Algorithm

In statistical mechanics we commonly want to evaluate thermodynamic averages of the form

$$\langle y \rangle = \frac{\sum_i y_i e^{-\beta E_i}}{\sum_i e^{-\beta E_i}} \tag{8}$$

where $E_i$ is the energy of the system in state $i$ and $\beta = 1/k_B T$ . Such problems can be solved using the Metropolis et al. (1953) algorithm.

Let us suppose the system is initially in a particular state and we change it to another state $j$. The detailed balance condition demands that in equilibrium the flow from $i$ to $j$ must be balanced by the flow from $j$ to $i$. This can be expressed as

$$p_i T_{i \to j} = p_j T_{j \to i} \tag{9}$$

where $P_i$ is the probability of finding the system in state $i$ and $T_{i\rightarrow j}$ is the probability (or rate) that a system in state $i$ will make a transition to state $j$. Eq.([?]) can be rearranged to read

$$\frac{T_{i\rightarrow j}}{T_{j\rightarrow i}} = \frac{p_j}{p_i} = e^{-\beta(E_j - E_i)}$$

Generally the right-hand-side of equation is known and we want to generate a set of states which obey the distribution $p_i$. This can be achieved by choosing the transition rates such that

$$T_{i\rightarrow j} = \begin{cases} 1 & if \quad p_j > p_i \quad or \quad E_j < E_i \\ e^{-\beta(E_j - E_i)} & if \quad p_j < p_i \quad or \quad E_j > E_i \end{cases}$$

In practice if $p_j < p_i$, a random number, $\tau$, is chosen between 0 and 1 and the system is moved to state $j$ only if the ratio is less than $p_j/p_i$ or $e^{-\beta(E_j - E_i)}$.

This method is not the only way in which the condition can be fulfilled, but it is by far the most commonly used.

An important feature of the procedure is that it is never necessary to evaluate the partition function, the denominator in (8) but only the relative probabilities of the different states. This is usually much easier to achieve as it only requires the calculation of the change of energy from one state to another.

Note that, although we have derived the algorithm in the context of thermodynamics, its use is by no means confined to that case.

## 29.1 Project: The Ising Model

### 29.1.1 Introduction

The Ising model for a ferromagnet is not only a very simple model which has a phase transition, but it can also be used to describe phase transitions in a whole range of other physical systems. The model is defined using the equation

$$H = -J \sum_{i,j_i} S_i S_j$$

where the i and j designate points on a lattice and $S_i$ takes the values $\pm 1/2$. The various different physical systems differ in the definition and sign of the various $J_{ij}$'s.

### 29.1.2 The Model and Method

Here we will consider the simple case of a 2 dimensional square lattice with interactions only between nearest neighbors. In this case

$$E = -J \sum_{i,j_i} S_i S_j$$

where $j_i$ is only summed over the 4 nearest neighbors of $i$.

This model can be studied using the Metropolis method as described in the notes, where the state can be changed by flipping a single spin. Note that the change in energy due to flipping the k$^{th}$ spin from $\uparrow$ to $\downarrow$ is given by

$$\Delta E_k = -J \sum_{j_k} S_{j_k}$$

The only quantity which actually occurs in the calculation is

$$Z_k = \exp(-\Delta E_k / k_B T)$$

and this can only take one of five different values given by the number of neighboring $\uparrow$ spins. Hence it is sensible to store these in a short array before starting the calculation. Note also that there is really only 1 parameter in the model, $J/k_B T$, so that it would make sense to write your program in terms of this single parameter rather than $J$ and $T$ separately.

The calculation should use periodic boundary conditions, in order to avoid spurious effects due to boundaries. There are several different ways to achieve this. One of the most efficient is to think of the system as a single line of spins wrapped round a torus. This way it is possible to avoid a lot of checking for the boundary. For an $N \times N$ system of spins define an array of $N^2$ elements using the shortest sensible variable type: char in C(++).

It is easier to use for spin $\uparrow$ and for spin $\downarrow$, as this makes the calculation of the number of neighboring $\uparrow$ spins easier. In order to map between spins in a 2d space $S_r$ $(r = (x, y))$ and in the 1d array $S_k$ the following mapping can be used.

$$\begin{aligned} S_{r+\delta x} &\longrightarrow S_{r+1} \\ S_{r-\delta x} &\longrightarrow S_{k+N^2-1} \\ S_{r+\delta y} &\longrightarrow S_{k+N} \\ S_{r-\delta y} &\longrightarrow S_{k+N^2-N} \end{aligned}$$

where the $2^{nd}$ $N^2$ elements of the array are always maintained equal to the $1^{st}$ $N^2$. This way it is never necessary to check whether one of the neighbors is over the edge. It is important to remember to change $S_{k+N^2}$ whenever $S_k$ is changed.

The calculation proceeds as follows:

1. Initialize the spins, either randomly or aligned.

2. Choose a spin to flip. It is better to choose a spin at random rather than systematically as systematic choices can lead to spurious temperature gradients across the system.

3. Decide whether to flip the spin by using the Metropolis condition (see notes).

4. If the spin is to be flipped, do so but remember to flip its mirror in the array.

5. Update the energy and magnetization.

6. Add the contributions to the required averages.

7. Return to step 2 and repeat.

### 29.1.3 The Physics

In general it is advisable to run the program for some time to allow it to reach equilibrium before trying to calculate any averages. Close to a phase transition it is often necessary to run for much longer to reach equilibrium. The behavior of the total energy during the run is usually a good guide to whether equilibrium has been reached. The total energy, E, can be calculated and the magnetization

$$M = \frac{1}{N} \sum_i S_i$$

It should be possible to calculate these as you go along, by accumulating the changes rather than by recalculating the complete sum after each step. A $10 \times 10$ lattice should suffice for most purposes and certainly for testing, but you may require a much bigger lattice close to a transition.

A useful trick is to use the final state at one temperature as the initial state for the next slightly different temperature. That way the system won't need so long to reach equilibrium.

It should be possible to calculate the specific heat and the magnetic susceptibility. The specific heat could be calculated by differentiating the energy with respect to temperature. This is a numerically questionable procedure however. Much better is to use the relationship

$$C_v \propto \frac{1}{N}\left(\frac{J}{k_B T}\right)^2 \left(\left\langle E^2 \right\rangle - \langle E \rangle^2\right) \tag{10}$$

Similarly, in the paramagnetic state, the susceptibility can be calculated using

$$\chi \propto \frac{1}{N}\frac{J}{k_B T}\left(\left\langle S^2 \right\rangle - \langle S \rangle^2\right) \tag{11}$$

where $S = \sum_i S_i$ and the averages are over different states, i.e. can be calculated by averaging over the different Metropolis steps. Both these quantities are expected to diverge at the transition, but the divergence will tend to be rounded off due to the small size of the system. Note however that the fact that ([**?**]) & ([**?**]) have the form of variances, and that these diverge at the transition, indicates that the average energy and magnetization will be subject to large fluctuations around the transition.

Mean field theory:

The Ising model

$$H = -\epsilon \sum_{\langle (i,j),(i',j') \rangle} J_{i,j} J_{i',j'} - B \sum_{(i,j)} J_{i,j}$$

For a specific configutaiom C, the probability is

$$p(C) = \frac{1}{Z(\epsilon, B)} \exp[-E_C/kT]$$

The partition function is

$$Z(\epsilon, B) = \sum_C \exp[-E_C/kT]$$

The internal energy:

$$E = \sum_C p(C) E_C = kT \frac{\partial}{\partial T} \ln Z(\epsilon, B)$$

The magnetization:

$$M = \sum_C p(C)(\sum_{(i,j)\in C} J_{i,j}) = kT\frac{\partial}{\partial B}\ln Z(\epsilon, B)$$

The specific heat:

$$C_B = \frac{\partial E}{\partial T} = \frac{1}{kT^2}\left\{\sum_C p(C)E_C^2 - E^2\right\}$$

The magnetic susceptibility:

$$\chi = \sum_C p(C)(\sum_{(i,j)\in C} J_{i,j}) = kT\frac{\partial}{\partial B}\ln Z(\epsilon, B)$$

# 30 Random Walks

One of the basic tasks of physics is to describe the motion of an object. For a single object, classical physics teaches us that, if we can identify all of the forces acting upon the object, then once we specify the position and velocity of the object at some reference time, we can use Newton's laws of motion to determine the position and velocity of the object at any previous or subsequent time. The difficulty, of course, is to identify all of the forces acting upon the object.

As the number of objects increase, the number of parameters that are required to describe the system, along with the number of forces acting upon any one object, increases. Eventually we reach a point where we are less concerned with the motion of a single object and want to look at the

system as a whole. This is the essence of statistical mechanics. Here we are interested in identifying those physical quantities that describe the state of the whole system. These quantities are, of course, completely determined by the aggregate; theoretically, if we specified the state of each object in the system at a particular time, we could completely specify the state of the system at that time. However, even for systems consisting of a couple hundred objects this becomes complicated due to the vector nature of most of the parameters involved.

The problem becomes even worse when we begin to ask about systems made up of molecules and atoms. For example, consider a gas of Hydrogen nuclei that contains $10^5$ atoms in a volume of 1.0 $m^3$. For each atom, we need to describe its initial position and velocity, each of which requires three numbers. This means that a total of $6 \times 10^5$ numbers are required already. Now we have to consider the effect of all of the forces acting on

any one atom. If all of the forces except for the electrostatic repulsion between the nuclei are ignored, then we only need to know the charge of the nuclei, which we can assume to be the same. Thus, we have to store $6 \times 10^5$ numbers to completely describe the system at any instant in time. Assuming that each number can be represented by an integer (and that an integer is 32 bits long), this means that $1.92 \times 10^7$ bits, or 2.4 MB, are needed just to store the positions and velocities. To calculate the forces, we would have to calculate and store the vector displacement between every possible pairing of atoms, which results in almost 5 billion combinations! So we see that a computer with $>$20 GB of memory is needed to compute the state of the system from first principles. But this gas is essentially a vacuum!

Even if we could design a computer with the memory and speed necessary, we have a serious flaw in our logic above, which makes this impossible.

If our gas consisted of beach balls, then we could use classical physics. Unfortunately, for a real gas made up of atoms, we can't use classical physics, but are instead forced to turn to quantum mechanics. In this case, we have a non-negligible effect from the Uncertainty Principle, which means that we cannot specify the initial conditions exactly to begin with, but instead can only state ranges over which the initial values will probably be found.

Given these problems, how can we go about describing the motion of a single particle in the overall conglomerate? To solve this, let's consider a specific example, namely a dust particle suspended in a glass of water. If we set up this as an actual experiment, we would soon observe that the dust particle does not remain stationary, but instead moves about. However, the motion is not constant, but rather consists of small, apparently random, movements. Applying our knowledge of physics, we know that this is

because the dust particle is colliding with the water molecules, rebounding off of them in a direction that conserves the momentum of each specific collision. Since we cannot see the water molecules, the direction of each rebound appears to be random.

Obviously, we cannot hope to predict the trajectory after even a small number of collisions. However, we can approximate the motion by assuming that the dust particle has an equal probability of moving in any direction. We can specify the model more exactly by appealing to the original formulation of the problem: "A drunk sailor is leaning against a lamp post and decides to head back to his ship. If he takes N steps of equal length in random directions, how far will the sailor be from the lamp post when he stops?" This is the basic idea of a random walk.

Random walks are very important in physics because their behavior is closely related to the solutions of the diffusion equation. In one dimension

the diffusion equation can be written as

$$\frac{\partial P(x,t)}{\partial t} = D\frac{\partial^2 P(x,t)}{\partial x^2},$$

where $D$ is the self-diffusion coefficient and $P(x,t)dx$ is the probability of a particle being in the interval between $x$ and $x+dx$ at time $t$. As an example, the probability $P(x,t)$ might represent the concentration of smoke molecules diffusing in the air. In three dimensions the second derivative $\partial^2/\partial x^2$ is replaced by the Laplacian $\nabla^2$.

## 30.1 Lattice Determination

Before beginning to study random walks, we need to understand how to describe the grid upon which the walk occurs. In reality, Brownian motion,

which is what the small scale, apparently random motion of particulate matter suspended in a fluid is called, traverses a continuous range of positions, but even on this tiny scale, it would take an infinite amount of memory to model this numerically. Since this is obviously impossible, we replace the continuous coordinate grid with a simpler finite lattice.

The easiest lattice is a one-dimensional lattice of equal sized steps. The scale of this lattice is specified by the step size, h, associated with the random walk. Notice that we used this to describe differentiation and integration, with the additional restriction that the displacement in the walk can only increase with time. If we relax this restriction, we can describe any random walk that uses a constant step size.

It is important to realize that the act of replacing our continuous number line with a lattice of quantized spatial locations also forces the time

displacement to be replaced with another discrete lattice. The two coordinates are still related to each other through the velocity, $h_i = v_{ij}\tau_j$, where h is the spatial step size and $\tau$ is the temporal step size.

Example:

A particle is constrained to move along a wire. It has an 50% chance of stepping to the right with a speed of 4.5 m/s and a 20% chance of stepping to the left with the same speed. How far has the particle moved after 5 minutes?

Let's first solve the problem analytically. On the average the total distance moved is given by $(\left\langle x^2 \right\rangle)^{1/2}$ . Notice that we use $x^2$ instead of $x$ because

we are only interested in the distance, not the displacement. This yields

$$\left(\left\langle x^2(N)\right\rangle\right)^{1/2} = \left(\left(\sum_{i=1}^{N} v\tau_i\right)^2\right)^{1/2} = (N)^{1/2} v\tau_i$$

We can also solve this problem numerically. Distance.c finds the average distance that the particle moves.

While a one-dimensional system is a useful learning tool, most systems of physical interest require two, three, four, or even more dimensions to properly describe the motion. Extending our lattice into these dimensions is easy – simply replace the single step during a time slice with N random numbers, where N is the dimensionality of the space, each of which represents the probability of a step in a particular direction along a single axis.

A more interesting situation arises when the velocity is no longer constant, such as when a net force is acting upon the system, for example. In this case, we are faced with three possible solutions to the problem. The first one is to keep the spacing between adjacent points on the lattice fixed and require the velocity to be rounded off or truncated in such a way that the step in any particular direction is always an integer multiple of the step size in that direction. This has many unsatisfactory aspects, and thus should be used only as a last resort. The second solution is to create two lattices, one on which the position is tracked, the other monitoring the velocity. While we would still have to round or truncate the position and velocity vectors to fit their individual lattices, having the velocity vector information allows corrections to be made to the position vector.

The third solution to the problem is to allow the lattice to resize itself. This is known as an adaptive mesh. While we could allow the individual cell size

to be rescaled with each step, this has the potential of becoming extremely consumptive of memory and computing time. What is more commonly done is to maintain a fixed cell size until some other parameter, usually the slope associated with the rate of change, exceeds a preprogramming value. At this point the cell (and its immediate neighbors, if desired) is repartitioned onto a smaller grid. If and when the parameter returns to the previous range, the tighter grid can be expanded out again.

## 30.2 Persistent Walks

The random walks described so far all share a common feature: they have no "memory" of the previous step when determining the current one. This

feature can be applied to a wide range of physical problems, but there are a number of other interesting problems for which this is not the case. In a persistent walk, the transition (or step) probability depends upon the previous transition.

Example:

A sailor is trying to walk up to the bridge on a ship caught in rough seas. If the deck is rolling through $30^0$ every minute (i.e., from level to a $30^0$ list to port, back to level, over to a $30^0$ list to starboard, and back to level again in one minute) and pitching through $20^0$ every two minutes, write a program that describes the sailor's most likely position after a user specified time interval.

Assume that both the pitch and the roll can be described by sinusoidal functions and that the effect of any walls, lifelines, bulkheads, etc., can be ignored. Then the roll can be approximated by

$$\theta = \theta_{\mathsf{max}} \sin \omega_R t$$

while the pitch becomes

$$\theta = \theta_{\mathsf{max}} \sin \omega_P t$$

Let the time interval be 1 second, and assume that the sailor can take one step of 0.25 m during this time. Also assume that the deck is level when the sailor is taking the first step. In order to take the fact that the sailor will often continue in the same direction once he starts moving, let the probability that he will move in the same direction be 75. The result is contained in sailor.c.

## 30.3 Restricted Walks

Another type of random walk is the restricted walk. In this case, certain points on the lattice are restricted, that is, if the object moves onto a point that is restricted it is removed from the simulation. Restricted walks can be used to model phenomena in condensed matter, among other things. For example, consider the following idealized model of energy transport in solids. The solid is represented as a lattice with two types of sites: hosts and traps. An incident photon is absorbed at a host site and excites the host molecule or atom. The excitation energy is transferred at random to one of the host's nearest neighbors and the original excited molecule returns to its ground state. The excitation energy travels through the lattice as a vibrational mode called the exciton. The exciton then walks through the lattice at random until it reaches a trap site. The exciton is then trapped and a chemical reaction occurs.

## 30.4 Variable Length Walks

It is also possible to perform random walks with steps that can vary in length. In this model the length a of each step is a random variable that has a probability density, P(a), and the direction of each step is uniformly random. For this type of walk, it is impossible to predefine a regular lattice. Instead, the lattice has to be built up as each step is calculated and linked together in a list. Also, since the probability density is no longer uniform, an inverse transformation must be used to generate step lengths which follow $P(a)$.

## 30.5 Serial correlation test

There are many ways that members of a sequence of random numbers are correlated with each other. The easiest one to test for is linear correlation. The primary aim here is to find out whether there is a tendency for each random number to be followed by another one of a particular type. For example, it will be quite unacceptable if a large random number is always followed by a small one (or a large one). As a check, we can define a correlation coefficient between a sequence of $n$ random numbers $X_1$, $X_2$,..., $X_n$, as

$$C = \frac{n(\sum_{i=1}^{n} X_{i-1}X_i) - (\sum_{i=1}^{n} X_i)^2}{n\sum_{i=1}^{n} X_i^2 - (\sum_{i=1}^{n} X_i)^2}$$

where $X_0$ in the first term of the numerator may be taken to be equal to $X_n$..

The coefficient C defined above is slightly different from other such coefficients. We are taking here a sum over products of the type $X_{i-1}X_i$, $X_iX_{i+1}$,$\cdots$. The value of C for an uncorrelated sequence is expected to be in the range $[\mu_n - 2\sigma_n, \mu_n + 2\sigma_n]$ for 95% of the time, with

$$\begin{aligned} \mu_n &= -\frac{1}{n-1} \\ \sigma_n &= \frac{1}{n-1}\frac{n+1}{n(n-3)} \end{aligned}$$

The resuls of a correlation test for the three diferent random number genera tors, a bad linear congrence generator a good lnear congruence generator and a subtraction generator are also listed in Table 7-1 In each case, a set of 5000 random numbers is used. The fact that resuls from linear congruence methods have a tendency to be correlated is reflected

by the relatively large absolute values of both the good and the bad generator. Shuffling among the random numbers destroys such short-range correlations as can be seen from the much smaller absolute values of C obtained. Furthermore, since the main diference between the good and the bad linear congruence generators is in the distribution of the random numbers produced the results of the two methods are fairy similar as far as the corelation test is concered. The value of C for the subtraction method is -0.003 in the test caried out essentialy a null value for a set of 5000 numbers. In fact, repeated tests with diferent sets of 5000 numbers, each set prduced by a diferent seed give values of C of roughly the same magnitude with both positive and negative signs.

Table 7-1: Tests of random number generators.

| Test | Frequency $\chi^2$ | Serial correlation $C$ | Run-up $\chi^2$ |
|---|---|---|---|
| Expected value | $\nu = 49$ | $[-0.028, +0.028]$ | $\nu \doteq 6$ |
| Linear congruence method | | | |
| Bad | 174 | $-0.022$ | 682 |
| After shuffling | 171 | 0.001 | 842 |
| Good | 41 | $-0.023$ | 4.7 |
| After shuffling | 44 | $-0.003$ | 2.8 |
| Subtraction method | 56 | $-0.003$ | 4.9 |

Figure 3: