Chapter 3 Monte Carlo Methods in Condensed Matter Physics

Monte Carlo method is one of the most widely used numerical methods in physics—from quantum field theory to condensed matter physics. In this Chapter, we first review the basics of Monte Carlo method (Ref. 1). We then go on for some applications of the Monte Carlo method in statistical physics problems (Ref. 2 and 3).

1. What Is Monte Carlo?

[Mostly from Kalos and Whitlock, *Monte Carlo Methods*]

1–1. Introduction

The name *Monte Carlo* was applied to a class of mathematical methods first by scientists working on the development of nuclear weapons in Los Alamos in the 1940s. The essence of the method is the invention of games of chance whose behavior and outcome can be used to study some interesting phenomena. While there is no essential link to computers, the effectiveness of numerical or simulated gambling as a serious scientific pursuit is enormously enhanced by the availability of modern digital computers.

It is interesting, and may strike some as remarkable, that carrying out games of chance or random sampling will produce anything worthwhile. Indeed some authors have claimed that Monte Carlo will never be a method of choice for other than rough estimates of numerical quantities.

Before asserting the contrary, we shall give a few examples of what we mean and do not mean by Monte Carlo calculations.

Consider a circle and its circumscribed square. The ratio of the area of the circle to the area of the square is $\pi/4$. It is plausible that if points were placed at random in the square, a fraction of $\pi/4$ would also lie inside the circle. If that is true (and we shall prove later that in a certain sense it is), then one could measure $\pi/4$ by putting a round cake pan with diameter L inside a square cake pan with side L and collecting rain in both. It is also possible to program a computer to generate random pairs of Cartesian coordinates to represent random points in the square and count the fraction that lie in the circle. This fraction as determined from many experiments should be close to $\pi/4$, and the fraction would be called an estimate for $\pi/4$. In 1,000,000 experiments it is very likely (95% chance) that the number of points inside the circle would range between 784,600 and 786,200, yielding estimates of $\pi/4$ that are between 0.7846 and 0.7862, compared with the true value of 0.785398...

The example illustrates that random sampling may be used to solve a mathematical problem, in this case, evaluation of a definite integral,

$$I = \int_0^1 \int_0^{\sqrt{1-x^2}} dx\, dy. \tag{1}$$

The answers obtained are statistical in nature and subject to the laws of chance. This aspect of Monte Carlo is a drawback, but not a fatal one since one can determine how accurate the answer is, and obtain a more accurate answer, if needed, by conducting more experiments. Sometimes, in spite of the random character of the answer, it is the most accurate answer that can be obtained for a given investment of computer times. The determination of the value of π can of course be done faster and more accurately by non-Monte Carlo methods. In many dimensions, however, Monte Carlo methods are often the only effective means of evaluating integrals.

A second and complementary example of a Monte Carlo calculation is one that S. Ulam cited in his autobiography. Suppose one wished to estimate the chances of winning at solitaire, assuming the deck is perfectly shuffled before laying out the cards. Once we have chosen a particular strategy for placing one pile of cards on another, the problem is a straightforward one in elementary probability theory. It is also a very tedious one. It would not be difficult to program a computer to randomize lists representing the 52 cards of a deck, prepare lists representing the different piles, and then simulate the playing of the game to completion. Observation over many repetitions would lead to a Monte Carlo estimate of the chance of success. This method would in fact be the easiest way of making any such estimate. We can regard the computer gambling as a faithful simulation of the real random process, namely, the card shuffling.

Random numbers are used in many ways associated with computers nowadays. These include, for example, computer games and generation of synthetic data for testing. These are of course interesting, but not what we consider Monte Carlo, since they do not produce numerical results. A definition of a Monte Carlo method would be one that involves deliberate use of random numbers in a calculation that has the structure of a stochastic process. By *stochastic process* we mean a sequence of states whose evolution is determined by random events. In a computer, these are generated by random numbers.

A distinction is sometimes made between simulation and Monte Carlo. In this view, simulation is a rather direct transcription into computing terms of a natural stochastic process (as in the example of solitaire). Monte Carlo, by contrast, is the solution by probabilistic methods of nonprobabilistic problems (as in the example of π). The distinction is somewhat useful, but often impossible to maintain. The emission of radiation from atoms and its interaction with matter is an example of a natural stochastic process since each event is to some degree unpredictable. It lends itself very well to a rather straightforward stochastic simulation. But the average behavior of such radiation can also be described by mathematical equations whose numerical solution can be obtained using Monte Carlo methods. Indeed the same computer code can be viewed simultaneously as a "natural simulation" or as a solution of the equations by random sampling. As we shall also see, the latter point of view is essential in formulating efficient schemes. The main point we wish to stress here is that the same techniques yield directly both powerful and expressive simulation and powerful and efficient numerical methods for a wide class of problems. We should like to return to the issue of whether Monte Carlo calculations are in fact worth carrying out. This can be answered in a pragmatic way: many people do them and they have become an accepted part of scientific practice in many fields. The reasons do not always depend on pure computational economy. As in our solitaire example, convenience, ease, directness, and expressiveness of the method are important assets, increasingly so as pure computational power becomes cheaper. In addition, as asserted in discussing π, Monte Carlo methods are in fact computationally effective, compared with deterministic methods when treating many dimensional problems. That is partly why their use is so widespread in operations research, in radiation transport (where problems in up to seven dimensions must be dealt with), and especially in statistical physics and chemistry (where systems of hundreds or thousands of particles can now be treated quite routinely). An exciting development of the past few years is the use of Monte Carlo methods to evaluate path integrals associated with field theories as in quantum chromodynamics.

1–2. A short history of Monte Carlo

Perhaps the earliest documented use of random sampling to find the solution of an integral is that of Comte de Buffon. In 1777 he described the following experiment. A needle of length L is thrown at random onto a horizontal plane ruled with straight lines a distance d ($d > L$) apart. What is the probability P that the needle will intersect one of these lines? Comte de Buffon performed the experiment of throwing the needle many times to determine P. He also carried out the mathematical analysis of the problem and showed that

$$P = \frac{2L}{\pi d}. \tag{2}$$

Some years later, Laplace suggested that this idea could be used to evaluate π from throws of the needle. This is indeed a Monte Carlo determination of π; however, as in the first example of this chapter, the rate of convergence is slow. It is very much in the spirit of inverting a probabilistic result to get a stochastic computation. We would call it an *analog* computation nowadays.

Lord Kelvin appears to have used random sampling to aid in evaluating some time integrals of the kinetic energy that appear in the kinetic theory of gases. His random sampling consisted of drawing numbered pieces of paper from a bowl. He worried about the bias introduced by insufficient mixing of the papers and by static electricity. W. S. Gossett used similar random sampling to assist in his discovery of the distribution of the correlation coefficient.

Many advances were being made in probability theory and the theory of random walks that would be used in the foundations of Monte Carlo theory. For example, Courant, Friedrichs, and Lewy showed the equivalence of the behavior of certain random walks to solutions of certain partial differential equations. In the 1930s Enrico Fermi made some numerical experiments that would now be called Monte Carlo calculations. In studying the behavior of the newly discovered neutron, he carried out sampling experiments about how a neutral particle might be expected to interact with condensed matter. These led to substantial physical insight and to the more analytical theory of neutron diffusion and transport.

During the Second World War, the bringing together of such people as Von Neumann, Fermi, Ulam, and Metropolis and the beginnings of modern digital computers gave a strong impetus to the advancement of Monte Carlo. In the late 1940s and early 50s there was a surge of interest. Papers appeared that described the new method and how it could be used to solve problems in statistical mechanics, radiation transport, economic modeling, and other fields. Unfortunately, the computers of the time were not really adequate to carry out more than pilot studies in many areas. The later growth of computer power made it possible to carry through more and more ambitious calculations and to learn from failures.

At the same time, theoretical advances and puting into practice powerful error-reduction methods meant that applications advanced far faster than implied by sheer computing speed and memory size. The two most influential developments of that kind were the improvements in methods for the transport equation, especially reliable methods of "importance sample" and the invention of the algorithm of Metropolis et al. The resulting successes have borne out the optimistic expectations of the pioneers of the 1940s.

2. Random Variables of Specified Distribution

In Chapter 2 we introduced methods of generating uniformly distributed random numbers. Very often, we want to have some other form of distribution.

2–1. Discrete distribution

Let's say that we want to generate a random variable i taking value 1 with probability p_1, value 2 with probability p_2, $\cdots$, value n with probability p_n. The sum of the probabilities is one,

$$\sum_{i=1}^{n} p_i = 1. \tag{3}$$

Let ξ be the uniformly distributed random number in the interval (0, 1). The random variable i distributed according to probability $p_1, p_2, \cdots, p_n$ can be generated by taking a random number ξ and deciding a value for i:

$$\begin{aligned}
&\text{if} \quad \xi \le p_1, \quad \text{then} \quad i = 1; \\
&\text{if} \quad p_1 < \xi \le p_1 + p_2, \quad \text{then} \quad i = 2; \\
&\cdots \\
&\text{if} \quad \sum_{k=1}^{m-1} p_k < \xi \le \sum_{k=1}^{m} p_k, \quad \text{then} \quad i = m.
\end{aligned}$$

The most common case is $n = 2$. A special case $p_1 = p_2 = \cdots = p_n = 1/n$ (the random variable i taking value 1 to n with equal probability) can be obtained with a simple operation

$$i = \lfloor n\,\xi \rfloor + 1, \tag{4}$$

where $\lfloor\ \rfloor$ means floor or truncation to integer.

Let's look at another example. We want to sample the geometric distribution

$$p_i = 2^{-i}, \qquad i = 1, 2, \ldots. \tag{5}$$

According to the above method, we should set $i = 1$ if $\xi < 1/2$; $i = 2$ if $1/2 < \xi < 1/2 + 1/4$; and in general $i = k$ if $\sum_{j=1}^{k-1} 2^{-j} < \xi < \sum_{j=1}^{k} 2^{-j}$. We can express the condition more concisely as

$$i = -\lfloor \ln \xi / \ln 2 \rfloor + 1. \tag{6}$$

The random variable i may also be generated by

```
f = 0.5;
i = 1;
r = drand48();
while( r < f) {
   f = 0.5 * f;
   ++i;
}
```

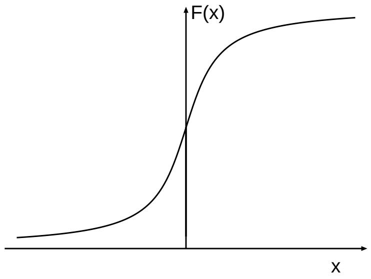
You may convence yourself that this indeed generates desired distribution. Since the probabilities decrease rapidly for large i, the chances are good that the loop terminates quickly. In fact two tests are required, on the average, before it does. The decision on which method to use to sample the geometric distribution depends on the computer used. If evaluations of logarithms are slow, it is better to use the second multiplicative method.

2–2. Continuous distribution

Consider the distribution of a single random variable x taking real values in some domain. Its probability density is $p(x)$. Probability density is defined such that the probability for random variable x taking values between x and $x+dx$ is $p(x)\,dx$. The distribution function $F(x)$ is defined by the probability that the random variable x is less than or equal to a given value x_0,

$$P(x \le x_0) = F(x_0) = \int_{-\infty}^{x_0} p(x)\,dx. \tag{7}$$

Since $F(x)$ is a probability, we have $0 \le F(x) \le 1$. The distribution function is a nondecreasing function of its argument.



The random variable x with probability density $p(x)$ can be generated with the formula

$$x = F^{-1}(\xi), \tag{8}$$

where $F^{-1}(\xi)$ is the inverse function of the distribution function $F(x)$, and ξ is a uniformly distributed random number.

Let's prove that x is indeed distributed according to $p(x)$. Since there is a one-to-one correspondence between x and ξ, the probability for x taking values between x and $x + dx$ is the same as for ξ between ξ and $\xi + d\xi$. Since ξ is uniformly distributed, the probability in this interval is just $d\xi$. And since $\xi = F(x)$, we have the probability for x between x and $x + dx$ as

$$1 \cdot d\xi = dF(x) = F'(x)\,dx = p(x)\,dx. \tag{9}$$

Example 1:

Generate x according to exponential distribution

$$p(x) = \begin{cases} e^{-x}, & x \ge 0; \\ 0, & x < 0. \end{cases} \tag{10}$$

The distribution function

$$F(x) = \int_{-\infty}^{x} p(x)\,dx = \int_{0}^{x} e^{-x}\,dx = 1 - e^{-x}, \quad x \ge 0. \tag{11}$$

The inverse function is

$$x = -\ln(1 - \xi). \tag{12}$$

You can also equivalently generate the exponential distribution with the formula $x = -\ln \xi$. Why?

Example 2:

Generate Gaussian distribution

$$p(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}, \quad -\infty < x < \infty. \tag{13}$$

It is helpful to generate a pair of Gaussian random numbers, since the inverse of the Gaussian distribution function (error function) can not be found analytically. The joint distribution of x and y is

$$p(x, y) = \frac{1}{2\pi} e^{-\frac{1}{2}(x^2+y^2)}, \quad -\infty < x, y < \infty. \tag{14}$$

Let's introduce polar coordinates $r = \sqrt{x^2 + y^2}$ and $\theta = \tan^{-1}(y/x)$, we can rewrite the probability density as

$$p(x, y)\,dx\,dy = \frac{1}{2\pi}\,e^{-\frac{1}{2}r^2}\,r\,dr\,d\theta. \tag{15}$$

From the above equation, we see that θ is distributed uniformly between 0 and 2π; and r is distributed according to $re^{-r^2/2}$. Thus the distribution function for r is

$$F_r(r) = \int_0^r re^{-\frac{1}{2}r^2} dr = 1 - e^{-\frac{1}{2}r^2} = \xi_1. \tag{16}$$

Solving for r

$$r = \sqrt{-2\ln(1 - \xi_1)}. \tag{17}$$

We can replace $1 - \xi_1$ by ξ_1 since it does not change the probability distribution. The random variable θ can be generated by $\theta = 2\pi\xi_2$. And finally, x and y can be generated by the formula

$$x = r\cos\theta = \sqrt{-2\ln\xi_1}\,\cos 2\pi\xi_2; \tag{18}$$

$$y = r\sin\theta = \sqrt{-2\ln\xi_1}\,\sin 2\pi\xi_2. \tag{19}$$

Here ξ_1 and ξ_2 are independent random numbers uniformly distributed between 0 and 1.

One can easily rescale the standard Gaussian distribution to get a Gaussian distribution with mean a and variance σ^2 by $y = \sigma x + a$.

Rejection method is a method which does not rely on the analytic expression of inverse distribution function. But it is less efficient. Assuming that $a \le x \le b$, and $0 \le p(x) \le c$, the number $x = a + \xi_1(b - a)$ is accepted if $c\xi_2 < p(x)$; otherwise a new value x is drawn and the condition is tested again.

The method of using the inverse distribution function is useful only if the inverse can be found easily. And it works only if one can reduce the problem to one random variable. In later sections, we'll study method (importance sampling) of generating arbitrary distributions involving many random variables. But we first review on numerical integration by Monte Carlo method.

3. Monte Carlo Evaluation of Finite-Dimensional Integrals

Let $X = (x_1, x_2, \cdots, x_d)$ be a point in a d-dimensional space, and $dX = dx_1 dx_2 \cdots dx_d$ the volume element. We want to evaluate the integral over the domain Ω,

$$G = \int_\Omega g(X)\,p(X)\,dX, \tag{20}$$

where $p(X)$ is interpreted as probability (density) satisfying

$$p(X) \ge 0, \quad \int_\Omega p(X)\,dX = 1. \tag{21}$$

The basic Monte Carlo method of calculating the integral is to draw a set (sequence) of random values X_1, X_2, $\cdots$, X_N distributed according to the probability $p(X)$, and to form the arithmetic mean

$$G_N = \frac{1}{N}\sum_{i=1}^{N} g(X_i). \tag{22}$$

The quantity G_N is approximately G, and we say G_N is an estimator of G. It is crucial to understand why $p(X_i)$ did not multiply $g(X_i)$. If X_i were equally spaced in the domain Ω, we would include $p(X_i)$. Since X_i is distributed according to $p(X_i)$—there are more points where $p(X_i)$ is large—we already take it into account. Very often the probability $p(X)$ itself is a uniform distribution $p(X) = \mathit{const}$ in the domain Ω. Other times, we need other form of distribution, e.g., in statistical mechanics, we like to have Boltzmann distribution, $p(X) \propto e^{-E/k_BT}$. Splitting the integrand $g(X)p(X)$ into a probability $p(X)$ and a quantity $g(X)$ is somewhat arbitrary and is for the convenience of computation. However, the choice of $p(X)$ does influence the statistical accuracy of the Monte Carlo estimates.

3–1. Fundamental theorems

We'll just state two important theorems which form the theoretical basis of the Monte Carlo method.

(1). **(Strong) Law of Large Numbers**: the arithmetic mean G_N converges with probability one to the expectation value G. In mathematical notation

$$P\{\lim_{N\to\infty} G_N = G\} = 1. \tag{23}$$

This theorem guarantees the convergence of Monte Carlo estimates to the correct answer in the limit of infinite many number of points. But it does not tell us how fast it converges. The next theorem tells us more.

(2). **Central Limit Theorem**: Let's define the variance as

$$\sigma^2 = \int_\Omega \big(g(X) - G\big)^2 p(X)\, dX = \langle g^2\rangle - \langle g\rangle^2, \tag{24}$$

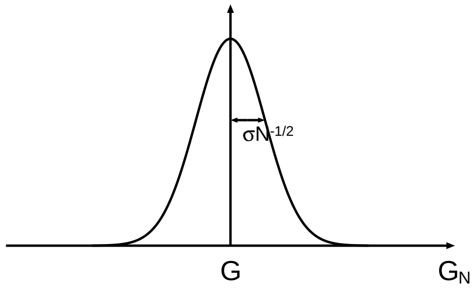
and a normalized value (also a random variable)

$$t_N = \frac{\sqrt{N}\,(G_N - G)}{\sigma}, \tag{25}$$

then

$$\lim_{N\to\infty} P\{a \le t_N < b\} = \int_a^b \frac{1}{\sqrt{2\pi}}\, e^{-\frac{1}{2}t^2}\, dt. \tag{26}$$

In other words, G_N is a random variable distributed according to Gaussian distribution with mean G and variance σ^2/N for sufficiently large N.



As $N \to \infty$, the observed G_N turns up in ever narrower interval near G and one can predict the probability of deviations measured in units of σ. The observed G_N is within one *standard error* (i.e. $\sigma/\sqrt{N}$) of G 68.3% of the time, within two standard errors 95.4% of the time, and within three standard errors 99.9% of the time. This is due to the property of the Gaussian distribution.

3–2. Error of Monte Carlo calculation

From the central limit theorem, we have

$$G = G_N + \text{error}. \tag{27}$$

The error of Monte Carlo estimates for G is

$$|\text{error}| = \epsilon \approx \frac{\sigma}{\sqrt{N}}. \tag{28}$$

From the definition of σ, we can also estimate the error ϵ as well as the value G itself. However, it is better to use the following *unbiased* estimator for σ^2.

$$\sigma^2 \approx \frac{N}{N-1}\left\{ \frac{1}{N}\sum_{i=1}^{N} g^2(X_i) - \Big(\frac{1}{N}\sum_{i=1}^{N} g(X_i)\Big)^2 \right\}. \tag{29}$$

Monte Carlo error decreases as $1/\sqrt{N}$, as the number of samples (points) N increases. Since N is proportional to the computer CPU time T, error $\epsilon \propto T^{-1/2}$. Let's compare the efficiency of Monte Carlo method with that of deterministic method of quadrature (e.g. trapezoidal or Simpson rule). The deterministic methods typically have error $\epsilon \propto h^k$ due to discretization, where h is grid size. The computer time goes as $T \propto h^{-d}$ in d dimensions when the grid size is decreased. Thus error goes as

$$\epsilon \propto \frac{1}{T^{k/d}}. \tag{30}$$

For Simpson rule of numerical integration, $k = 4$. We see that for a one-dimensional integral, the error decreases much faster than that of Monte Carlo method. However as the dimensions of the integral increase, the Simpson rule becomes worse than the Monte Carlo method for a fixed amount of computer time.

In conclusion, standard numerical quadrature is very good for low dimensions. Monte Carlo method is superior for higher dimensional integrations.

3–3. Some examples

Example 1:

Evaluate the integral $\int_0^1 \sqrt{1-x^2}\,dx = \pi/4$.

Let's take the probability distribution as

$$p(x) = \begin{cases} 1, & 0 \le x \le 1; \\ 0, & \text{otherwise}, \end{cases} \tag{31}$$

so that x is uniformly distributed in the unit interval. And

$$g(x) = \sqrt{1-x^2}. \tag{32}$$

we can cast the integral in the required form $\int g(x)\,p(x)\,dx$. The Monte Carlo estimator for the integral is

$$G_N = \frac{1}{N}\sum_{i=1}^{N} g(x_i) = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(1-x_i^2)}.$$

This can be implemented in C with a few lines

```
sum = 0;
for(i = 0; i < N; ++i)
   sum += sqrt(1 - pow(drand48(), 2));
mean = sum / N;
```

Example 2:

Evaluate the integral $\int_0^\infty e^{-x}\cos x\,dx$, and estimate the Monte Carlo error.

Since the integral domain is from 0 to ∞, it is impractical to sample the value x uniformly from 0 to ∞. However, it is quite natural to sample x from 0 to ∞ by the exponential distribution, $p(x) = e^{-x}$, and then $g(x) = \cos(x)$. Even though x can take any positive value, but the probability that it takes large value is very small. We sample the exponential distribution by $x = -\ln\xi$, where ξ is a random number uniformly distributed between 0 and 1. A Monte Carlo estimator for the integral is then

$$G_N = \frac{1}{N}\sum_{i=1}^{N}\cos(\ln\xi_i). \tag{33}$$

Why do we omit the exponential factor? We can also estimate the error $\sigma/\sqrt{N}$ in G_N by a Monte Carlo calculation of σ.

$$\sigma^2 \approx \frac{N}{N-1}\left\{\frac{1}{N}\sum_{i=1}^{N}\cos^2(\ln\xi_i) - G_N^2\right\}. \tag{34}$$

4. Importance Sampling (Metropolis Algorithm)

In the Monte Carlo calculation of integration outlined in the above section, we assumed that we could generate X with probability $P(X)$. We have studied methods of generating one-dimensional distribution by inverting the distribution function and by rejection. However, transformation method works only if one can invert the function analytically, and the rejection method is inefficient. In higher dimensions, we need other methods of generating the distribution. Importance sampling is the method of generating probability distribution efficiently on a computer.

4–1. Markov chain

A Markov chain is a mathematical notion for a random walk in the space of X. Remember that $X = (x_1, x_2, \cdots, x_d)$ is a point in a d-dimensional space. We'll sometimes refer X as a state of the system.

We start from some initial point X_0, the next point X_1 is generated according to certain transition probability. In this way a sequence of random points $X_0, X_1, \cdots, X_N$ is generated. These points are required to appear with the probability $P(X)$. The rule to generate next point X_{i+1} given the point X_i is described by the transition probability

$$P(X_{i+1}|X_i) = W(X_i \to X_{i+1}). \tag{35}$$

where $W(X_i \to X_{i+1})$ is called transition probability from state X_i to X_{i+1}, and $P(X_{i+1}|X_i)$ is the probability that the system is in state X_{i+1} given that the system was in state X_i, namely, it is the conditional probability. We have

$$W(X_i \to X_{i+1}) \geq 0, \quad \sum_{X_{i+1}} W(X_i \to X_{i+1}) = 1. \tag{36}$$

Since W is probability, it satisfies the usual constraint of probability—probability must be positive and total probability adds to one. The method of importance sampling is to choose a proper transition probability W in such a way so that the distribution $P(X)$ is the desired distribution we wanted to generate. More precisely, let $P_0(X)$ be some probability distribution, which is the initial distribution of the states. The new distribution after one step of move is then

$$P_1(X) = \sum_{X'} P_0(X')W(X' \to X). \tag{37}$$

In general we have

$$P_k(X) = \sum_{X'} P_{k-1}(X')W(X' \to X). \tag{38}$$

The theorems of stochastic processes tell us that the limit for large k exists and is unique (with certain conditions)

$$\lim_{k\to\infty} P_k(X) \to P(X). \tag{39}$$

And $P(X)$ satisfies the equation

$$P(X) = \sum_{X'} P(X')W(X' \to X). \tag{40}$$

We say that $P(X)$ is invariant under the transition of W, or $P(X)$ is the equilibrium distribution for the given transition probability.

To have a better, intuitive understanding of the above equations, you may think of the probability as population of certain particles. Imagine that the particles can be in a number of (quantum-mechanical) states denoted by X. Initially at time 0, the particles are distributed according to $P_0(X)$. Each particle has a tendency to go to other states. For example, if the particle is at state X', the probability that it will go to state X is $W(X' \to X)$. Thus, we have (number of particles times the probability) $P_0(X')W(X' \to X)$ number of particles going to state X. After a unit time, the population of particles at state X, $P_1(X)$, is the sum of the number of particles going from any state X' to the state X. This transitions of particles will eventually make the population steady in any states—we have reached the equilibrium.

In a computer simulation, typically only one state X is stored in memory. The transition changes the state. The distribution of the state is also given by $P_k(X)$. By this we mean that if we do the same simulation many times starting with X_0 distributed according to $P_0(X)$, the outcome after k steps will be distributed according to $P_k(X)$.

The sequence of points X_i generated by the random walk is correlated, since the next point is generated based on the knowledge of the previous point. This is in contrast with the simple sampling method, e.g., the inverse distribution function method, where the sequence of points is uncorrelated. Each point is generated afresh. The penalty due to the correlation is that the statistical error will be larger than the formula $\sigma/\sqrt{N}$, which is valid only if the points are uncorrelated.

4–2. Ergodicity

We mentioned the existence and uniqueness of the equilibrium distribution. But we slipped over the condition for it. The condition is the following: starting from any state X, it is possible to get to any another state X' after a finite number of transitions. We call such Markov chain ergodic (or regular, irreducible). We can view the transition probability W as a matrix. Let $W^n(X, X')$ be the matrix element of the n-th power of W. The ergodic condition means that the matrix element at every entry (X, X') is nonzero for some value n.

4–3. Detailed balance

How to choose W to get prescribed P? It is sufficient that W satisfies

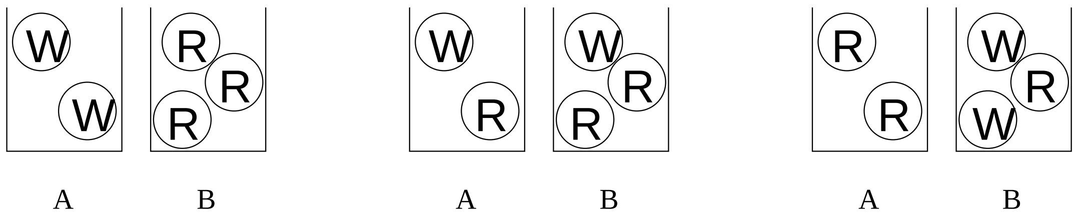
$$P(X')W(X' \to X) = P(X)W(X \to X'), \tag{41}$$

where P is the known distribution. If W satisfies this equation and it is ergodic, then the Markov chain will generate X according to the probability distribution $P(X)$. It is possible that W does not satisfy the detailed balance condition but still generates the distribution P. That's why we say that the above condition is a sufficient condition.

Let's prove that the distribution is invariant under the transition W satisfying the detailed balance condition. Summing over the variable X' and using the normalization condition for W, we have

$$\sum_{X'} P(X')W(X' \to X) = \sum_{X'} P(X)W(X \to X') = P(X)\sum_{X'} W(X \to X') = P(X). \tag{42}$$

Example:



Consider two pots, A and B, with three red balls and two white balls distributed between them so that A always has two balls and B always has three balls. The balls of same color are not distinguishable. There are three different configurations for the pots as shown above, corresponding to three states X_1, X_2, X_3. We obtain transitions between these three configurations by picking a ball out of A at random and one out of B at random and interchanging them. Inspection shows that we can make the following transitions from state i to j with probability $W(i \to j)$, $W(1 \to 1) = W(1 \to 3) = 0$, $W(1 \to 2) = 1$, $W(2 \to 1) = 1/6$, $W(2 \to 2) = 1/2$, $W(2 \to 3) = 1/3$, $W(3 \to 1) = 0$, $W(3 \to 2) = 2/3$, and $W(3 \to 3) = 1/3$. The transition probabilities in matrix form is

$$W = \begin{pmatrix} 0 & 1 & 0 \\ 1/6 & 1/2 & 1/3 \\ 0 & 2/3 & 1/3 \end{pmatrix}. \tag{43}$$

Since

$$W^2 = \begin{pmatrix} 1/6 & 1/2 & 1/3 \\ 1/12 & 23/36 & 10/36 \\ 1/9 & 5/9 & 1/3 \end{pmatrix}, \tag{44}$$

all the elements are nonzero, the transition matrix is regular and ergodic, we expect to find a unique stationary probability distribution P_i, independent of our initial state. The probabilities P_i satisfy the equation (an eigenvalue problem with eigenvalue 1)

$$P_i = \sum_{j=1}^{3} P_j W(j \to i). \tag{45}$$

In matrix notation, we have $P = PW$, or $W^T P^T = P^T$, where $P = (P_1, P_2, P_3)$, and T for transpose. Together with the normalization condition $P_1 + P_2 + P_3 = 1$, the solution to the set of linear equation is

$$P_1 = \frac{1}{10}, \quad P_2 = \frac{6}{10}, \quad P_3 = \frac{3}{10}. \tag{46}$$

This is the probability distribution after many transitions. And we have lost all knowledge of the initial state.

We can say this in another way. Let us assume that, initially, the system is in the first configuration. We are certain of this. The initial state (our knowledge) is $P_1^0 = 1$, $P_2^0 = P_3^0 = 0$. Now we perform k transitions (k large). What configuration is the system in? We do not know! The configurations are distributed according to P_i and will remain that way. There is a 10% chance the system is in configuration 1, a 60% chance to be in configuration 2, and a 30% chance to be in 3. If we could perform the experiment many times, that is, starting from configuration 1 and making k transitions and then looking at the configuration, we would find and 10% of the time it would end up in configuration 1, 60% of the time it would end up in configuration 2, and 30% of the time it would end up in configuration 3. But for each experiment, we could never be certain of the outcome.

We can see how fast the system converges to the equilibrium probability distribution. Noting that if $P^0 = (1, 0, 0)$,

$$
\begin{aligned}
P^1 =& P^0 W = (0, 1, 0), \\
P^2 =& P^1 W = P^0 W^2 = (\frac{1}{6}, \frac{1}{2}, \frac{1}{3}), \\
P^3 =& P^2 W = P^0 W^3 = (\frac{1}{12}, \frac{23}{36}, \frac{5}{8}) \approx (0.083, 0.638, 0.277), \\
P^4 =& P^3 W = P^0 W^4 = (\frac{23}{216}, \frac{127}{216}, \frac{11}{36}) \approx (0.106, 0.587, 0.3), \\
p^6 =& P^0 W^6 \approx (0.1007, 0.5986, 0.3007).
\end{aligned}
$$

We see that after six transitions, the probabilities are already very close to the limits.

4–4. Metropolis algorithm

An importance Markov chain Monte Carlo sampling for a specific distribution $P(X)$ goes as follows: starting from some initial state (configuration) X_0, a sequence of states will be generated by a Markov chain through the transition probability $W(X_i \to X_{i+1})$. The starting state may be a unique state or may be generated at random with certain distribution P_0. The new state X_{i+1} is based on the old state X_i. The new state is generated in such a way so that the probability that the system is in state X_{i+1} is just $W(X_i \to X_{i+1})$, knowing that the system was in state X_i. Hopefully we can generate this probability distribution easily. Otherwise we might have generated $P(X)$ directly in the first place. It is sufficient that W satisfies detailed balance condition and the Markov chain is ergodic. Metropolis algorithm refers to a specific choice of W:

$$W(X \to X') = T(X \to X') \min\left(1, \frac{P(X')}{P(X)}\right), \quad X \neq X', \quad T(X \to X') = T(X' \to X). \tag{47}$$

Matrix T can be any distribution but must be symmetric. We almost always use

$$T(X \to X') = \begin{cases} const, & \text{for } X' \text{ inside certain region around } X; \\ 0, & \text{otherwise.} \end{cases} \tag{48}$$

The probability that the state does not change, $W(X \to X)$, is determined by the normalization condition

$$\sum_{X'} W(X \to X') = 1. \tag{49}$$

Let's describe more concretely the algorithm.

Metropolis Algorithm: Generate a sequence X_0, X_1, X_2, $\cdots$, X_i, $\cdots$, with a probability distribution $P(X_i)$.

1) Specify an initial state X_0, set $i = 0$.
2) Generate a new state $\overline{X}_{i+1}$ according to $T(X_i \to \overline{X}_{i+1})$.
3) Accept the new state with probability $\min\bigl(1, P(\overline{X}_{i+1})/P(X_i)\bigr)$. That is, $X_{i+1} = \overline{X}_{i+1}$ if a random number between 0 and 1 is less than $P(\overline{X}_{i+1})/P(X_i)$; retain the old state as (i+1)-th state, $X_{i+1} = X_i$, otherwise.
4) set $i = i + 1$, go to 2.

Expectation values (or averages) can be estimated by

$$\langle A(X) \rangle = \sum_X A(X) P(X) \approx \frac{1}{N} \sum_{i=M+1}^{M+N} A(X_i). \tag{50}$$

It is important that we throw away the first M points in the average. We have to wait for the system approaching equilibrium before we use the points. The reason for this is that the system is distributed according to $P_k(X)$, and we want the distribution $P(X)$. $P_k(X)$ will converge to $P(X)$ for sufficiently large

k. This convergence is usually fast. For efficiency consideration, usually we do not use every X_i even if we have reached equilibrium, since they are highly correlated. We calculate $A(X_i)$ only every certain number of iterations or steps.

5. Examples of Metropolis Algorithm

5–1. Gaussian distribution

We have seen how to generate Gaussian distribution by some transformation. We can also generate Gaussian distribution by a Metropolis algorithm. This can be done as follows: starting with some arbitrary real value x, we then propose a new value as

$$\bar{x} = x + \alpha\Big(\xi - \frac{1}{2}\Big), \tag{51}$$

where ξ is a uniformly distributed random number between 0 and 1; α is some constant, say, 0.3. The new proposed value $\bar{x}$ is distributed uniformly inside an interval of length α, centered at the original value x. This choice of $\bar{x}$ is equivalent to take a particular functional form of $T(x \to \bar{x})$. As long as T is symmetric, the exact form of T does not matter as far as generating correct distribution is concerned. The value $\bar{x}$ will be accepted as the next point if a uniformly distributed random number between 0 and 1 is less than $\exp\big[-\frac{1}{2}(\bar{x}^2 - x^2)\big]$, otherwise, the old value x is taken as the next point. This generates a sequence of points, which should be distributed according to Gaussian with zero mean and unit variance.

5–2. Ising Model for Magnets

Ising model is the simplest model for ferromagnets. One of the most important feature of a magnet is that it has spontaneous magnetization at lower temperatures. However, the magnetization disappears at some higher temperature T_c. The phase below T_c is called ferromagnetic phase and that above T_c is called paramagnetic phase. T_c is called critical temperature and there is a phase transition at this temperature. This situation is very much like that in percolation.

Let's take two-dimensional system as an example. On an $L \times L$ square lattice, we put a spin σ_i at each lattice site i. The spin here is a "classical" spin, not the quantum-mechanical operator. The spin can be thought as the z-component of the Pauli spin, taken only two values $+1$ and -1. For each configuration denoted by $\{\sigma\} = (\sigma_1, \sigma_2, \sigma_3, \ldots)$ (the collection of all the spins), the system has a definite energy given by

$$H\big(\{\sigma\}\big) = -J \sum_{\langle i,\, j\rangle} \sigma_i \sigma_j - B \sum_i \sigma_i, \tag{52}$$

where J is called coupling constant; B is proportional to the external magnetic field. The first summation runs over the nearest neighbors only, i.e., each bond is summed once. Such model is also referred as nearest neighbor Ising model. The second summation is over all the lattice sites. When J is positive, the model describes a ferromagnet; while $J < 0$ describes an anti-ferromagnet. Let's consider just one term in the nearest neighbor interaction. The interaction energy of neighbors takes only two values, $+J$ or $-J$. For ferromagnet, the energy is lower if the two spins are parallel. It costs energy if the spins are pointing in different directions.

According to statistical-mechanical description, the system will not be in a definite state. Due to thermal fluctuation, the system will have some probability distribution among the states. If the temperature T is a fixed parameter, we have the famous Boltzmann distribution (Canonical distribution)

$$P\big(\{\sigma\}\big) \propto \exp\left(-\frac{H(\{\sigma\})}{k_B T}\right), \tag{53}$$

where $k_B \approx 1.38 \times 10^{-23}$ Joule/Kelvin is called the Boltzmann constant. Our goal is to calculate average quantities such as energy, magnetization, correlation functions, etc. The internal energy per spin is defined by

$$u = \frac{1}{L^d}\langle H\rangle = \frac{1}{L^d}\sum_{\{\sigma\}} H\big(\{\sigma\}\big)P\big(\{\sigma\}\big); \tag{54}$$

and the magnetization is defined as

$$m = \frac{1}{L^d}\langle M\rangle = \frac{1}{L^d}\sum_{\{\sigma\}}\Big|\sum_i \sigma_i\Big|P\big(\{\sigma\}\big). \tag{55}$$

The summation is over all possible states. Since each site can have two states (spin up and spin down), we have 2^{L^d} states for a system of linear size L in d dimensions.

Few other quantities are useful in Ising model simulation. The specific heat $c = du/dT$ and susceptibility $\chi = dm/dB$ can be calculated from the fluctuations

$$c = \frac{1}{k_B T^2 L^d}\Big[\langle H^2\rangle - \langle H\rangle^2\Big]; \tag{56}$$

$$k_B T L^d \chi = \begin{cases} \langle M^2\rangle - \langle M\rangle^2, & T < T_c; \\ \langle M^2\rangle, & T > T_c. \end{cases} \tag{57}$$

Here the angular brackets $\langle\cdots\rangle$ denote average over the Boltzmann distribution. And $M = |\sum_i \sigma_i|$ is the absolute value of the total magnetization of a particular state. The fourth-order cumulant

$$g = \frac{1}{2}\left[3 - \frac{\langle M^4\rangle}{\langle M^2\rangle^2}\right], \tag{58}$$

is useful in determining the critical temperature.

A Monte Carlo simulation of the Ising model involves the following steps:

0) Initialize σ_i at each site with arbitrary spin values, e.g., all spin up, or up or down with equal probability. The complete set of spins $\{\sigma\}$ is the abstract state X discussed in the previous sections.

1) Choose a site i at random and propose to flip the spin at that site, $\sigma'_i = -\sigma_i$. The proposed state $\overline{X}$ is the one with one spin at location i flipped. This amounts to take

$$T(X \to \overline{X}) = \begin{cases} 1/L^d, & \text{if } X \text{ and } \overline{X} \text{ differ by 1 spin;} \\ 0, & \text{otherwise.} \end{cases} \tag{59}$$

2) Calculate the energy increment

$$\Delta E = H\big(\{\sigma'\}\big) - H\big(\{\sigma\}\big) = 2J\sigma_i \sum_{\substack{\text{neighbors} \\ \text{of } i}} \sigma_j + 2B\sigma_i. \tag{60}$$

3) Accept the proposed state as the new state if a uniformly distributed random number (between 0 and 1) is less than $\exp(-\Delta E/k_B T)$; retain the old state as the new state otherwise.

4) go to 1.

One Monte Carlo step will be defined as performing the above basic single-spin flip L^d number of times. After one Monte Carlo step, each site on average has tried once to flip. You can also go through the lattice in sequence, and try to flip each spin at the site. But choosing sites at random seems better.

The following is an implementation of the two-dimensional Ising model without the magnetic field ($B = 0$). This function performs one Monte Carlo step by trying to flip spins L^2 times. It picks a site at random and tries to flip it. The flip is actually performed if the energy change is negative or zero, or if the random number is less than the probability $\exp(-\Delta E/k_B T)$ when the energy change is positive. `N` is the total number of spins, `Z` is coordination number. Spin `s[ ]` and temperature `T` are passed globally.

```
#include <math.h>                    exponential function and random number need this
                                                                   macro definitions
#define  L  16                                                     lattice linear size
#define  N  (L*L)                                              total number of spins
#define  Z  4                                                   coordination number
int s[N];                                                               spin +1 or –1
double  T = 2.26918531;                                      dimensionless temperature

void monte_carlo()
{
   int i, j, k, e;                                                 i is the center site
   int nn[Z];                                                  the names of neighbors

   for(k = 0; k < N; ++k) {                                     do one Monte Carlo step
      i = drand48() * (double) N;                         pick the center site at random
      neighbor(i, nn);                                             find neighbors of site i
```

```
        for(e = 0, j = 0; j < Z; ++j)             go over the neighbors
           e += s[nn[j]];                         sum of the neighbor spins
        e *= 2 * s[i];                            2 times the center spin
        if (e <= 0)                               when energy change is less than
           s[i] = - s[i];                         or equal to zero, spin is flipped
        else if (drand48() < exp(-e/T))           otherwise, it is flipped
           s[i] = - s[i];                         with probability less than one
     }
  }
```

The `neighbor()` function is the same as in the previous Chapter for the cluster labelling algorithm.

6. Analysis of Ising Model Monte Carlo data

We'll first introduce what is known about Ising model and then present general picture of second-order phase transitions. We then discuss how to analyze the data obtained from Monte Carlo simulation.

6–1. Exact results of two-dimensional Ising model

The exact solution of the Ising model in two dimensions is a great victory of statistical mechanics, solved by Onsager in 1944. The mathematics for the solution is extremely involved. We'll just state out the result.

(1) The critical temperature T_c is determined by the equation

$$\sinh \frac{2J}{k_B T_c} = 1, \quad \text{or equivalently} \quad \tanh \frac{J}{k_B T_c} = \sqrt{2} - 1, \quad \text{or} \quad e^{-2J/k_B T_c} = \sqrt{2} - 1. \tag{61}$$

Solving one of these equations, the dimensionless temperature has the value $k_B T_c/J \approx 2.269$.

(2) The average energy per site (total energy divided by the number of spins) in zero magnetic field ($B = 0$) is

$$u = -J \left\{ 1 \pm \frac{2}{\pi} \sqrt{1 - z^2}\, I(z) \right\} \coth 2K, \quad K = \frac{J}{k_B T}, \tag{62}$$

where the $+$ sign is for $T < T_c$ and $-$ sign is for $T > T_c$, and

$$I(z) = \int_0^{\pi/2} \frac{d\omega}{\sqrt{1 - z^2 \sin^2 \omega}}, \quad z = \frac{2 \sinh 2K}{\cosh^2 2K}. \tag{63}$$

(3) The derivative of u with respect to T is the specific heat. Near the critical point it is asymptotically

$$\frac{c}{k_B} \approx \frac{2}{\pi} \left(\ln \coth \frac{\pi}{8} \right)^2 \left[\ln \left(\frac{\sqrt{2}}{|K - K_c|} \right) - 1 - \frac{\pi}{4} \right]. \tag{64}$$

(4) The spontaneous magnetization ($B \to 0^+$) per site is

$$m = \left(1 - \frac{1}{\sinh^4 2K} \right)^{1/8} \approx (T_c - T)^{1/8}, \quad T < T_c. \tag{65}$$

This last result is first obtained by C. N. Yang. Of course $m = 0$ for $T \geq T_c$.

6–2. General description of critical behavior and critical exponents

Ising model in two and higher dimensions has a phase transition from ferromagnetic phase to paramagnetic phase. The transition is called continuous or second-order phase transition. The behavior near the critical temperature T_c is very similar to that in percolation, i.e., many quantities obey a power law. The magnetization behaves as

$$m \propto (T_c - T)^{\beta}, \quad T < T_c,\ T \to T_c, \tag{66}$$

and the specific heat diverges

$$c \propto |T - T_c|^{-\alpha}, \quad T \to T_c. \tag{67}$$

The susceptibility also diverges

$$\chi \propto |T - T_c|^{-\gamma}, \quad T \to T_c. \tag{68}$$

For the two-dimensional Ising model, the critical exponents are $\beta = 1/8$, $\alpha = 0$, and $\gamma = 7/4$. The specific heat exponent $\alpha = 0$ is consistent with $c \propto \ln |T - T_c|$.

6–3. Finite-size scaling

The theory and analytical results outlined above apply to infinitely large system. Or in the language of statistical mechanics, they are the results of thermodynamic limit. However, a computer simulation is always performed on finite lattices. This is okay in most of the range of parameters of the model. The effects of finite sizes show up when the physics gets interesting—near the critical point finite-size effect becomes important.

In fact, the finite-size effect is not necessarily a drawback. It can be utilized to determine the ratios of critical exponents. According to finite-size scaling theory, the values of thermodynamic quantities at the critical point vary with sizes as follows

$$c(T_c) \propto L^{\alpha/\nu}, \quad m(T_c) \propto L^{-\beta/\nu}, \quad \chi(T_c) \propto L^{\gamma/\nu}, \tag{69}$$

where the exponent ν is called correlation length exponent ($\nu = 1$ for the two-dimensional Ising model). The specific heat and susceptibility have peaks near the critical point. The location of the peak depends on the system size. Finite-size scaling theory predicts that the peak will shift according to

$$T_{peak}(L) - T_c \propto L^{-1/\nu}. \tag{70}$$

This is one way to determine the location of the critical temperature T_c as well as the exponent ν. However, because that we have three fitting parameters (T_c, ν, and proportionality constant) it is not easy to get accurate value for ν.

Finite-size scaling theory gives a general description of the temperature and size dependencies of the thermodynamic quantities. In general, if T is the only variable in the thermodynamic limit, then a quantity, say, the susceptibility, has the following finite-size dependence

$$\chi(T, L) = L^{\gamma/\nu} \tilde{\chi}\big((T - T_c)L^{1/\nu}\big). \tag{71}$$

The function $\tilde{\chi}(x)$ is called (universal) scaling function. The finite-size dependence of the fourth-order cumulant is particularly interesting:

$$g(T, L) = \tilde{g}\big((T - T_c)L^{1/\nu}\big). \tag{72}$$

It becomes size independent at T_c. Below T_c it increases with sizes and above T_c it decreases with sizes. We can use this property to determine T_c.

7. Critical Slowing Down

If one compares the simulation of percolation and that of Ising model, the simulation of percolation is computationally more efficient in that each configuration generated is independent of the previous configuration. In Ising model simulation, or more generally any simulation using the Metropolis algorithm, the next configuration depends on the previous one. Due to this correlation between Monte Carlo steps, the formula for the statistical error $\epsilon \approx \sigma/\sqrt{N}$ underestimates the true error. The error formula should be replaced by

$$\epsilon \approx \sigma \sqrt{\frac{1 + 2\tau}{N}}. \tag{73}$$

That is, the error is larger by a factor $\sqrt{1 + 2\tau}$ than the uncorrelated data. The quantity τ is called correlation time, which is roughly the number of Monte Carlo steps needed to generate independent configurations. We can compute the correlation time τ as follows: define the time-dependent correlation function,

$$f(t) = \frac{\langle A(t')A(t' + t)\rangle - \langle A\rangle^2}{\langle A^2\rangle - \langle A\rangle^2}, \tag{74}$$

where time t is measured in terms of Monte Carlo steps—time is passed by one unit when one updates the system by one Monte Carlo step. And $A(t')$ is the quantity at steps t' whose error we want to estimate. The correlation time is defined by

$$\tau = \sum_{i=1}^{\infty} f(t). \tag{75}$$

In an actual numerical estimate of τ we can not sum upto infinity due to statistical errors in $f(t)$. Typically we truncate the sum or extrapolate the correlation function to infinity.

Now we are ready to explain the concept of critical slowing down. Just like many other quantities near the critical point, the correlation time also becomes singular at the critical point. On an infinite lattice, it obeys a power-law

$$\tau \propto |T - T_c|^{-\nu z}. \tag{76}$$

Here ν is the correlation length exponent and z is called dynamic critical exponent. This phenomena that the correlation time becomes very large is called critical slowing down. It not only happens in computer simulation but also appears in real experiments. On a finite system of linear size L, the correlation time will, of course, not go to infinity. But it will grow with size as

$$\tau \propto L^z, \quad T = T_c. \tag{77}$$

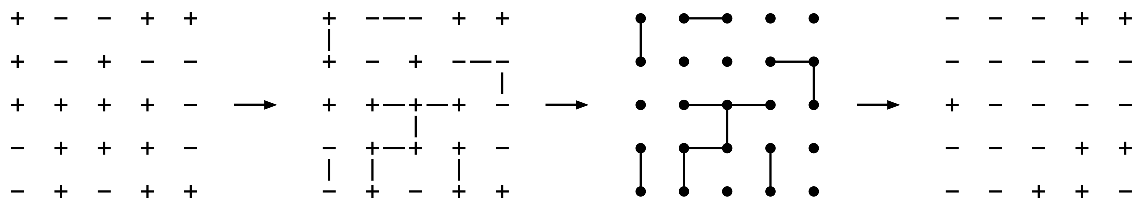
Substituting this result into the error formula, we find that $\epsilon \approx \sigma L^{z/2}/N^{1/2}$ for large system near T_c. For two-dimensional Ising model with Metropolis algorithm, $z \approx 2.1$, we see typically that increasing the size leads to a larger error in the quantity to be calculated.

8. Cluster Monte Carlo algorithms

The Metropolis algorithm makes changes locally one site at a time. This is the cause of the critical slowing down. There are algorithms that change the states by a group of clusters. The advantage of the cluster algorithms is that they are faster, not necessarily in computer time per Monte Carlo step, but in terms of dynamics. That is, it has a much smaller value of z.

8–1. Swendsen-Wang algorithm

The Swendsen-Wang multi-cluster algorithm is closely related to percolation. One starts with a spin configuration $\{\sigma\}$ and generates a percolation configuration based on the spin configuration by the rule described below. Then the old spin configuration is forgotten and a new spin configuration $\{\sigma'\}$ is generated based on the percolation configuration. The rule is such that the detailed balance is satisfied. Or more generally, the transition leaves the equilibrium probability invariant. The following is the algorithm for nearest neighbor Ising model without magnetic field ($B = 0$). It can be generalized to other models and with magnetic field.



Swendsen-Wang Algorithm:

0) Start with some arbitrary state $\{\sigma\}$.

1) Go through each nearest neighbor connection of the lattice, create a bond between the two neighboring sites i and j with probability $p = 1 - e^{-2J/(k_B T)}$, only if the spins are the same, $\sigma_i = \sigma_j$. Never put a bond between the sites if the spin values are different. We are creating a bond percolation configuration with probability p on a subset of lattice sites where the spins are either all pointing up or down.

2) Identify clusters as a set of sites connected by bonds, or isolated sites. Two sites are said to be in the same cluster if there is a connected path of bonds joining them. Every site has to belong to one of the clusters. After the clusters are found, each cluster is assigned a new Ising spin chosen with equal probability between $+1$ and -1. The old spin values now can be ‘forgotten’. And all the sites in a cluster now take the value of spin assigned to the cluster.

3) One Monte Carlo step is finished. Repeat 1 for the next step.

The performance of the algorithm in terms of correlation time in comparison with Metropolis algorithm is remarkable. Recall that the dynamic critical exponent $z \approx 2$ for Metropolis algorithm almost independent of the dimensionality. The Swendsen-Wang algorithm in one dimension gives $z = 0$ (τ approaches a constant as $T \to 0$). For the two-dimensional Ising model, the dynamic critical exponent z is less than 0.3 or possibly zero (but $\tau \propto \ln L$). In three dimensions it is about 0.5. At and above four dimensions it is 1.

8–2. Wolff single-cluster algorithm

In the Swendsen-Wang algorithm, we generated many clusters and then flip these clusters. In Wolff algorithm one picks a site at random, and then generates one single cluster by growing a cluster from the seed. The neighbors of the seed site will also belong to the cluster if the spins are in parallel and a random number is less than $p = 1 - e^{-2J/(k_B T)}$. That is, the neighboring site will be in the same cluster as the seed site with probability p if the spins have the same sign. If the spins are different, neighboring site will never belong to the cluster. Neighbors of each new site in the cluster are tested for membership. This testing of membership is performed on pair of sites (forming a nearest neighbor bond) not more than once. The recursive process will eventually terminate. The spins in the cluster are turned over with probability 1. Single-cluster algorithm appears more efficient than the multi-cluster one.

The following is a fairly efficient way of implementing the Wolff algorithm. Read very carefully. Make sure that you understand the program. The `monte_carlo_step(n)` function does the Wolff single-cluster flip `n` times.

```
#include <math.h>                          drand48() need this
                                           macro definitions
#define  L  16                             lattice linear size
#define  N  (L*L)                          total number of spins
#define  Z  4                              coordination number = 2 · d
                                           global variables
int s[N];                                  spin +1 or -1
double  p = 0.5;                           percolation probability p = 1 - e^{-2J/(k_B T)}

void monte_carlo_steps(int n)
{
   int i, k;

   for(k = 0; k < n; ++k) {                do the Wolff single-cluster flip n times
      i = drand48() * (double) N;          pick a seed site at random
      flip(i, s[i]);                       flip a cluster starting from the seed
   }
}
```

The next function is the core part which performs a Wolff single-cluster flip. This function is recursive. The array for spins `s[ ]`, percolation probability `p`, and coordination number `Z` are passed globally. The first argument `i` of the `flip()` function is the site to be flipped, the second argument `s0` is the spin of the cluster before flipping. The function `neighbor()` is used again as in single-spin-flip Ising model or percolation program.

```
void flip(int i, int s0)
{
   int j, nn[Z];

   s[i] = - s0;                            flip the spin immediately
   neighbor(i, nn);                        find nearest neighbor of i
   for(j = 0; j < Z; ++j)                  flip the neighbor if
      if(s0 == s[nn[j]] && drand48() < p)  spins are equal and
         flip(nn[j], s0);                  random number is smaller than p
}
```

9. Other Applications of the Metropolis Algorithm

Metropolis algorithm and similar algorithms have wide applications in physics. Examples are quantum system and quantum field theory, fluids, macro-molecules and polymer systems. In this section, we'll describe briefly how to simulation fluid and lattice model of polymer systems.

9–1. Simple fluid

Let's imagine that we have a (two-dimensional) box of dimension $L \times L$. Inside, we put certain number of particles. Depending on the density of the particles, we might get a fluid state or a solid state. We assume that the particles can be described as classical point particles. The particles interact with each other via a potential energy

$$V(r) = 4\,\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]. \tag{78}$$

This form of interacting energy is called Lennard-Jones potential. It is a very good model for noble gases like Argon or Krypton. The parameter ϵ is the depth of the potential well, and σ can be thought as roughly the size of the atoms. For Argon the two parameters have the value $\sigma \approx 3.5\,\text{Å}$ and $\epsilon/k_B \approx 118\,\text{K}$.

We are interested in the calculation of the thermodynamic quantities like internal energy, specific heat, equation of state (pressure as a function of temperature and volume), entropy, etc. A Monte Carlo sampling generates the particle configurations according to the Boltzmann distribution $P \propto \exp(-E/k_BT)$. This can be achieved by the Metropolis algorithm. In fact, the Metropolis algorithm was just designed for such problem in 1953, by N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (ref. 4) on an Los Alamos MANIAC computer. This algorithm has been used widely since then.

To perform a Monte Carlo move, one picks a particle at random and try to move it inside certain small region, say a square of side α with coordinates changing to

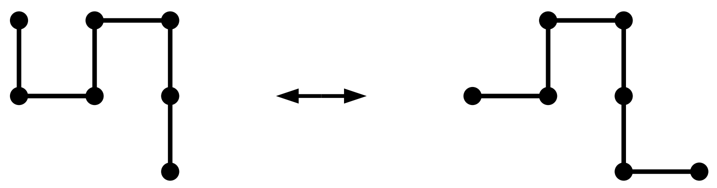
$$x \to x + \alpha\Big(\xi_1 - \frac{1}{2}\Big), \quad y \to y + \alpha\Big(\xi_2 - \frac{1}{2}\Big), \tag{79}$$

where ξ_1 and ξ_2 are two independent uniformly distributed random numbers between 0 and 1. We then calculate the change in energy of the system ΔE, due to this move. If $\Delta E \le 0$, i.e., if the move would bring the system to a state of lower energy, we allow the move and put the particle in the new location. If $\Delta E > 0$, we allow the move with probability $\exp(-\Delta E/k_BT)$; i.e., we take a random number ξ between 0 and 1, and if $\xi < \exp(-\Delta E/k_BT)$, we move the particle to the new position. Otherwise we return to its old position. Note, however, whether we moved the particle or not, we always count it as one Monte Carlo step. Once we have an equilibrium distribution, it is possible to calculate the equation of state by virial theorem (relating pressure with certain average of force). We'll discuss virial theorem in the next Chapter.

9–2. Simulation of polymer melt

Polymers are more complex than simple fluids. Again as before, we try to model the system as simple as possible. The lattice polymer models have been studied extensively in the literature. To represent a polymer system, we take a square lattice (or cubic lattice in three dimensions) and occupy a set of N sites connected by bonds, forming a one-dimensional chain. The sites or bonds can not be occupied more than once. Each single occupied site is called a monomer, which is supposed to represent an atom or a group of atoms. Monomers of the same polymer or different polymers will have interactions. Usually, it is taken as nearest neighbor interactions.

There are a number of Monte Carlo simulation methods for polymers. Probably the easiest to implement is the reptation method, also called slithering snake algorithm. We try to move a polymer by erasing a monomer at one end of the chain and attaching it to the other end. Again, we use the Metropolis method to decide whether we accept the move. If the total energy of the system decreases or stays the same, we allow the move. If the energy increases by an amount ΔE, we make the move only with a probability $\exp(-\Delta E/k_BT)$. Unsuccessful trial of move is also counted as a move (or Monte Carlo step). The polymer simulation algorithms can be used to study many properties of polymer systems.



10. Other Applications of Monte Carlo Method

We have talked about Monte Carlo methods mostly for numerical calculation of multi-dimensional integrations. But the methods are more general. It is possible to solve the integral equation

$$\chi(X) = S(X) + \int T(X,Y)\,\chi(Y)\,dY, \tag{80}$$

by Monte Carlo method under certain conditions. Some partial differential equations lead naturally to Monte Carlo solutions, e.g., diffusion equation by random walkers. Schrödinger equation can be solved by a method called Green function Monte Carlo. We shall not go into these areas. Interested reader may consult the book by Kalos and Whitlock.

References

1. M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods, Vol. I: Basics*, (John Wiley & Sons, 1986).
2. K. Binder and D. W. Heermann, *Monte Carlo Simulation in Statistical Physics – An Introduction*, 2nd ed., (Springer-Verlag, 1992).
3. D. W. Heermann, *Computer Simulation Methods in Theoretical Physics*, 2nd ed., (Springer-Verlag, 1990).
4. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, *J. Chem. Phys.* **21**, 1087 (1953).

TUTORIAL PROBLEM SET 3.

Questions 1–3, Wed, 28 February; questions 4 and 6, Wed, 13 March, 1996

1. Random Variables

(a) A random variable ξ with a uniform probability distribution in the interval $(0,1)$ is given. Work out a method to generate a random integer $k \geq 0$ (from ξ) with the probability distribution

$$P_k = (1-\alpha)\,\alpha^k, \quad k = 0, 1, 2, \cdots,$$

where α is some constant satisfies $0 < \alpha < 1$.

(b) Find a transformation to generate a random variable x (from a uniformly distributed random number $0 < \xi < 1$) with the distribution

$$p(x) = \frac{2}{\pi}\frac{1}{1+x^2}, \quad 0 < x < \infty.$$

2. Random Walker

Consider a random walker confined on three sites: left, middle, and right. If he's at the middle, he can walk to the left or right with equal probability; if he's at left or right, he can only walk to the middle. View this process as a Markov chain, what is the transition matrix W? What is the equilibrium distribution P of the walker? More challenging problem: what is the result if the walker can walk on a one-dimensional lattice of length L?

3. MONTE CARLO METHOD, MARKOV CHAIN

Consider a situation where we have only three possible states 1, 2, and 3 (e.g. discrete quantum states). The transition probability is given as $W(1 \to 2) = 1$, $W(2 \to 1) = W(2 \to 3) = W(3 \to 2) = W(3 \to 3) = \frac{1}{2}$, and $W(1 \to 3) = W(3 \to 1) = W(1 \to 1) = W(2 \to 2) = 0$. We can write in matrix form

$$W = \begin{pmatrix} 0 & 1 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}.$$

(a) Is the Markov chain specified by the above transition probabilities ergodic?

(b) Calculate the equilibrium probability distribution P_i, $i = 1, 2, 3$.

(c) Does the equilibrium distribution satisfy detailed balance condition?

$$P_i\, W(i \to j) = P_j\, W(j \to i), \qquad i, j = 1, 2, 3.$$

4. EXACT RESULTS OF ONE-DIMENSIONAL ISING MODEL

Calculate the partition function, free energy, internal energy and heat capacity of the one-dimensional Ising model without magnetic field. The Hamiltonian of the model is

$$H = -J \sum_{i=1}^{L} \sigma_i \sigma_{i+1}.$$

Use periodic boundary condition so that site $L+1$ is the same as site 1. *Hint*: break up the terms and write the partition function as matrix multiplications.

5. DETAILED BALANCE

Show that the Metropolis transition rate $W(X \to X') = T(X \to X') \min\big(1, P(X')/P(X)\big)$ satisfies detailed balance condition.

6. GLAUBER ALGORITHM AND HEAT BATH ALGORITHM

Consider Ising model again. The transition probability is not unique. There are many choices. In the Metropolis algorithm, the spin is flipped with probability $\min\big[1,\, \exp(-\Delta E/k_B T)\big]$. We'll show that the detailed balance is also satisfied if:

(a) (Glauber dynamics) Flip the spin with probability

$$\frac{1}{2}\left[1 - \sigma_i \tanh\Big(\frac{J}{k_B T} \sum_{\langle i,\, j\rangle} \sigma_j\Big)\right],$$

where site i is the center site whose spin is to be flipped; the summation is over the nearest neighbors of the center site.

(b) (Heat-bath algorithm) Reset of the center spin as

$$\sigma_i = \mathrm{sign}\left(\xi - \frac{e^{-\Delta}}{1 + e^{-\Delta}}\right), \qquad \Delta = \frac{2J}{k_B T} \sum_{\langle i,\, j\rangle} \sigma_j,$$

where ξ is a uniformly distributed random number between 0 and 1.

LABORATORY EXERCISE 3.

Due Saturday, 16 March, 1996

1. MONTE CARLO EVALUATION OF INTEGRALS

The ideal gas law,

$$\frac{P}{\rho k_B T} = 1, \quad \rho = \frac{N}{V},$$

applies when the size of the particles can be neglected and particles move freely without interactions. This is the case if the gas is very dilute. As a better approximation, we may assume that the particles are hard spheres of diameter σ, moving like billiard balls. When the size and the interaction of the particles are taking into account, the equation of state is modified as

$$\frac{P}{\rho k_B T} = 1 + b_2\rho + b_3\rho^2 + b_4\rho^3 + \cdots,$$

where b_2 is called second virial coefficient, b_3 third virial coefficient, etc. The virial coefficients are multi-dimensional integrals which we want to evaluate by Monte Carlo method in this exercise. From Mayer cluster expansion theory, we have

$$b_2 = -\frac{1}{2}\int d^3\mathbf{r}_1\, f(\mathbf{r}_1 - \mathbf{r}_0),$$

$$b_3 = -\frac{1}{3}\int\!\!\int d^3\mathbf{r}_1\, d^3\mathbf{r}_2\, f(\mathbf{r}_1 - \mathbf{r}_0)\, f(\mathbf{r}_2 - \mathbf{r}_0)\, f(\mathbf{r}_2 - \mathbf{r}_1),$$

$$\begin{aligned} b_4 = -\frac{1}{8}\int\!\!\int\!\!\int d^3\mathbf{r}_1\, d^3\mathbf{r}_2\, d^3\mathbf{r}_3 \Big[& 3 f(\mathbf{r}_1 - \mathbf{r}_0)\, f(\mathbf{r}_2 - \mathbf{r}_1)\, f(\mathbf{r}_3 - \mathbf{r}_2)\, f(\mathbf{r}_3 - \mathbf{r}_0) + \\ & 6 f(\mathbf{r}_1 - \mathbf{r}_0)\, f(\mathbf{r}_2 - \mathbf{r}_1)\, f(\mathbf{r}_3 - \mathbf{r}_2)\, f(\mathbf{r}_3 - \mathbf{r}_0)\, f(\mathbf{r}_3 - \mathbf{r}_1) + \\ & f(\mathbf{r}_1 - \mathbf{r}_0)\, f(\mathbf{r}_2 - \mathbf{r}_1)\, f(\mathbf{r}_3 - \mathbf{r}_2)\, f(\mathbf{r}_3 - \mathbf{r}_0)\, f(\mathbf{r}_3 - \mathbf{r}_1)\, f(\mathbf{r}_2 - \mathbf{r}_0)\Big], \end{aligned}$$

where $\mathbf{r}_0 = (0,0,0)$, $\mathbf{r}_i$ is three-dimensional vector and the notation $d^3\mathbf{r}_i$ stands for $dx_i\, dy_i\, dz_i$. Integration domain is over the whole space. The function f is defined as

$$f(\mathbf{r}_i - \mathbf{r}_j) = \begin{cases} -1, & \text{if } |\mathbf{r}_i - \mathbf{r}_j| < \sigma; \\ 0, & \text{otherwise.} \end{cases}$$

Note that b_2 can be evaluated easily by hand; b_3 is complicated, you really need a Monte Carlo evaluation. And b_4 is even more complicated. It is possible to write down higher order terms b_5, b_6, $\cdots$, but you can imagine that they will be extremely long. Try to evaluate the first three virial coefficients and also estimate the Monte Carlo errors. (To check your program, the answers should be $b_2 = \frac{2}{3}\pi\sigma^3$, $b_3/b_2^2 = 5/8$, $b_4/b_2^3 \approx 0.2869495$.)

2. METROPOLIS AND WOLFF ALGORITHMS

In this exercise, we like to write two versions of Monte Carlo simulation programs (standard single-spin-flip algorithm and Wolff's cluster algorithm) for ferromagnetic Potts model in two dimensions. Potts model is a generalization of the Ising model. An Ising spin takes two values $+1$ and -1. A q-state Potts spin takes q different values. The energy of a particular Potts state is

$$H\big(\{\sigma\}\big) = -J\sum_{\langle i,j\rangle} \delta_{\sigma_i\sigma_j}, \quad \sigma_i = 1, 2, \cdots, q.$$

Here the summation is over each nearest-neighbor bond once. $\delta_{\sigma_i\sigma_j}$ is the Kronecker δ function, i.e.,

$$\delta_{\sigma_i\sigma_j} = \begin{cases} 1, & \text{if } \sigma_i = \sigma_j, \\ 0, & \text{otherwise.} \end{cases}$$

For $q = 2$, this model is equivalent to Ising model. Please do the following:

(a) Write a straightforward single-spin-flip Metropolis algorithm for the Potts model (for arbitrary integer $q > 1$). You may modify my programs for your work. The purpose of this program is for comparison and debugging of the next program.

(b) Write a Wolff single-cluster algorithm for the same problem. Make sure you obtain the same result (within statistical error) for energy and specific heat at the same system size L and temperature T as by the program of (a).

(c) Calculate [using Wolff algorithm of (b)] and present a curve of specific heat c v.s. T for $q = 3$, $L = 16$, taking 10 to 20 data points in an interesting region of T. Draw data points as well as error bars. Give a rough estimate of T_c from this curve.

3. READING THE CLASSIC

Read ref. 4, the paper Metropolis *et al* proposed their method and studied the two-dimensional hard disk problem. Scribble down your notes, not more than a page.