Applications of Monte Carlo Methods to Statistical Physics

Austin Howard & Chris Wohlgamuth

April 28, 2009
An Introduction to the Monte Carlo Method

Perspective: Calculating Integrals

Suppose you were given the assignment of calculating an integral like the following:

\[ \int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x, y) \, dx \, dy \quad (1) \]

How would you go about doing it numerically? We learn several algorithms in Calculus for calculating integrals numerically. For example, we could use the midpoint method, where we subdivide the region of integration into finite “cells,” and take the sum over all cells of \( f \) evaluated in the middle of that cell multiplied by the cell’s volume. This provides a reasonably accurate approximation to the integral.

The problem is in the amount of calculation involved. In our two dimensional example above, if we divide each “axis” into \( n \) segments, we will have to perform a total of \( n^2 \) function evaluations. In a three dimensional integral, we will have to perform a total of \( n^3 \) function evaluations. In phase space (with 3 space and 3 momentum variables) we would have \( n^6 \) evaluations. In other words, the number of evaluations (which determines the length of time of our calculation) increases exponentially with the number of degrees of freedom which we are integrating over. With this in mind, think of some of the integrals we need to calculate in Statistical Mechanics problems (to find the partition function, etc.). We have 3 space and 3 momentum variables for every particle! Clearly, in order to perform any practical calculation (involving millions or billions of degrees of freedom), and to finish in any reasonable length of time, we need a strategy whose length of calculation does not grow exponentially as we add more particles.

Random Sampling: The Monte Carlo Class of Algorithms

In the 1940’s and 50’s, at Los Alamos National Lab, new forms of numerical calculations were being developed. While the specific algorithm varied from one calculation to another, all involved a particular technique: instead of systematically probing the domain of possible values in a grid-like fashion, they randomly selected (“sampled”) points from within the domain. This led to far less computational time (which was especially important during the early days of computing), and, if the problems were “well behaved” enough, did not significantly decrease the accuracy of the computation, as long as a reasonably large number of iterations of the “sampling” were performed. The practice of randomly sampling these “scenarios,” or “configurations,” came to be known as a Monte Carlo (MC) Method.\(^1\) Evidently, MC simulations do not come in a well defined equation or package. Instead, the MC method can better be thought of as a process or systematic approach. In this report we aim to supply the reader with a generalization of how one could go about constructing their own MC simulation and specific examples of the MC method in Statistical Physics.

\(^1\)The method is named after the famous Monte Carlo casino in Monaco, a small city-state near France and Italy.
Recursion Relations versus Markov Chains

There are many ways we can construct our random selections. The simplest way would be to “directly sample” our domain using a set of randomly generated numbers. One way of constructing a set of random numbers is by a recursion formula. We start with an integer, \( R_1 \), and generate successive integers by the recursion,

\[ R_{i+1} = a \times R_i + b \mod m \]  

(2)

where \( a, b, \) and \( m \) are integers. The variable \( a \) is called the generator or multiplier, \( b \) the increment, and \( m \) is the modulus. Start by choosing an integer \( R_1 \) between the values of 0 and \( m - 1 \). Calculate \( a \times R_i + b \), divide the result by \( m \), and find the remainder calling it \( R_2 \). Repeat this process \( m - 1 \) times.

However, there are certain situations in which this “direct sampling” approach proves quite impractical, because a given randomly selected configuration may prove to violate physical laws, and will have to be skipped over. For example, suppose we were modeling the behavior of several hard spheres bouncing around in a box. One random sample in this case would involve using our random number generator many times to choose the initial positions and momenta of all of our spheres.\(^2\) Sometimes our choice would place two spheres close enough together that they would overlap. There is no reason why this would not be selected by our random number generator, but it is obviously not a valid initial configuration. Hence, in our algorithm for this problem, we would have to add a mechanism to check, each time we take a random sample, whether or not it is a valid configuration (whether or not any spheres overlap). This makes the code less efficient for two reasons: it adds the “checking” overhead to every iteration, and the time to construct and test all of the invalid samples is wasted.\(^3\)

Another way, and the solution to our problem above, is by using a variant of the MC Method that uses Markov Chains. In a Markov Chain, the \( n + 1 \) iteration is not chosen randomly, as it was before, but is a specific function of the \( n^{th} \) configuration. Moreover, the \( n + 1 \) configuration is only dependent on the \( n^{th} \) configuration, and is independent of all prior states. With a clever enough choice of this function, we can eliminate these “invalid” configurations. While a detailed discussion of Markov Chain Monte Carlo Methods is beyond the scope of this paper, suffice it to say that, in practice, they are the most commonly used type of MC Simulations.

Percolation

The first example we will use to illustrate the application of MC to Statistical Physics is percolation. Percolation theory is the study of the diffusion of a liquid through a porous material. We use MC to answer the following general question:

\(^2\)Even though there is repeated use of a random number generator, it is still one sample.

\(^3\)The number of rejected configurations varies depending on the problem, but can be quite high (imagine many spheres in a small box; most configurations have overlap).
For a given material, with a specific porosity, will the liquid be able to percolate completely through the material, and if so, how much?

As with any computer simulation, we need to develop a mathematical model of the situation that can easily be implemented on a computer. Consider this model: Let us regard our porous material as an \( n \times n \times n \) grid of points (called nodes) connected by line segments (called bonds) to their nearest neighbors. Each of these line segments is then randomly given the property of being open or closed. If a line segment is open, the liquid is able to “pass through” that segment; if it’s closed, the liquid is blocked along that route. Let’s also say that at each assignment, the probability we make the line segment open is \( p \), that we make it closed is \( 1 - p \). This is a concise, and not too overly simplified, model of percolation; \( p \) is analogous to the porosity of our original material. Furthermore, the answer to our question is especially easy: the liquid will be able to percolate through the material if and only if there exists a “path” from one face of the cube to the opposite one which consists of only open line segments. We thus ask: for a given \( p \), what is the probability \( P(p) \) of such a path existing?

One way we could approach this problem is as follows: We create an array, each element of which contains a binary value. Then, using our random number generator, we go through the array and, one by one, assign a value of true or false (1 or 0) to each element. This corresponds to assigning the property or open of closed to each bond.\(^4\) Then we look, one by one, at a node on one “surface” and trace all possible paths through the matrix along true values. The proportion of “spanning” paths to total paths is precisely the probability \( P(p) \). While this is very straightforward, unfortunately, this system cannot work in practice: there are far too many computations that have to be done! First, we have to trace all possible paths from one node on the surface. Then we have to repeat for every one of the nodes. Also, in order to use the MC method, we need many, many “runs” with the same probability, so we must repeat the whole process a number of times with the same value of \( p \). Finally, in order to get the percolation \( P(p) \) as a function of \( p \), we must repeat all of this many times for different values of \( p \). The result is that this method is far too inefficient to work in practice.

Our next attempt to solve this problem is called the Hoshen-Kopelman Algorithm (HKA). It improves upon the previous algorithm in several ways. The main change is that we consider our material in a different light: instead of considering a matrix of the bonds, we perform the calculations with a matrix of the nodes. This reduces the size of our data by \( \frac{2}{3} \), and it allows HKA to be applied. The general goal of HKA is to label clusters of adjoining elements of a matrix which have the same value. In our application of the algorithm, the matrix\(^5\) (call it \( N \) for Nodes) contains only values of 1 or 0 (occupied or unoccupied). The matrix is created in the same way as before, randomly assigning each element the value of 1 or 0 with probability \( p \) and \( 1 - p \), respectively. HKA labels each cluster with a different positive integer, and stores the result in another matrix (call it \( L \) for labels). While the details of HKA are not too complicated, they do take some time to explain. In the interest of brevity, we will restrict ourselves to an overview of this technique.

\(^4\)Note that this array will be larger than \( n \times n \times n \), since there are 3 times more bonds than nodes.

\(^5\)In this case, the matrix \( \text{will be } n \times n \times n \).
The Hoshen-Kopelman Algorithm loops through all the elements of \( N \). When it encounters an element with value 1 (say at position \([n, m]\)) , it checks to see if the element above it \((N_{n-1, m})\) is also 1. If so, it assigns \( L_{n, m} \) the value of \( L_{n-1, m} \). It does the same for the element to the left (at \([n, m-1]\)). If neither of those elements \((N_{n-1, m} \text{ and } N_{n, m-1})\) are 1, then it assigns \( L_{n, m} \) a new label.

What happens when HKA encounters an element \([n, m]\) where \( N_{n-1, m} \text{ and } N_{n, m-1} \) are both 1, but \( L \) is different at those two points? We have found that what we thought were two different clusters are actually part of the same cluster! We solve this problem by the use of a third storage area (call it \( E \) for “equal labels”). HKA assigns the value\(^6\) in \( L_{n-1, m} \) to \( L_{n, m} \), but it also “makes a note” in \( E \) that the values in \( L_{n-1, m} \) and \( L_{n, m-1} \) actually represent the same cluster.

Once HKA has looped through all elements of \( N \), it goes back to \( E \), and determines how many distinct clusters there really are. Then it eliminates the “redundancy” in \( L \) using this information. Then \( L \) would be in its final form.

**Results of the Monte Carlo Method Applied to Percolation**

After the above mathematics and computation, the physical results one obtains are quite intriguing. The following is a graph of \( P(p) \) (normalized so that \( P(1) = 1 \)).\(^7\)

![Graph of P(p)](image)

A few comments about this graph:

\(^6\)This could just as well be \( L_{n-1, m} \); it is just a matter of convention.

\(^7\)This graph and the following one were made using computer code written by one of us (ARH), with portions based on work released under the GNU Public License by Tobin Fricke of The University of California at Berkeley. Because of this, we are obliged to release the code under the same license (see the References section).
1. This simulation was done for a 2D matrix (500 × 500). For each value of \( p \), 5000 matrices were randomly generated to test.

2. The limiting behavior \( (p \to 0, p \to 1) \) is as we would expect.

3. Note that the graph exhibits an abrupt change at \( p_c \approx 0.6 \). For all \( p < p_c \), there is no percolation whatsoever, while for \( p > p_c \), there is. This “critical probability” is something that is known from percolation theory. In fact, it is quite reminiscent of a phase transition; percolation is one of the simplest physical models to have phases.

Another interesting relation, while more of a mathematical curiosity, is: for a given \( p \), how many clusters (of size larger than 1 node) are generated within the material? We can plot that as well, and we get the following:

![Graph showing the number of clusters vs. p](image)

This fits our expectations as well: smaller values of \( p \) create many more small clusters within the material (but not too small where almost every cluster consists of just one particle).

**Ising Model**

A system of many interacting particles can exhibit a phase transition: that is, exhibit a change in a feature that characterizes the system. A ferromagnet exhibits a permanent macroscopic magnetization whereas a paramagnet exhibits some magnetic properties, but not a macroscopic magnetization. At the critical temperature a ferromagnetic system changes from one displaying macroscopic magnetization into a seemingly unmagnetized system, a paramagnetic system. With suitable approximations, predictions can be made from such small systems that can be used to predict the properties of a larger system in that same
region. A simple model with magnetic spins to show magnetic phase transitions is needed. The Ising Model assumes that each spin is able to point in only an upward or downward direction, and that the $i^{th}$ spin in the system only has the value of $\pm s_i$. There are interactive forces between the spins, and the Ising Model assumes that the interaction between the nearest neighbors is the only interaction in the system. The spins interact in pairs and have different energies when the spins are aligned and when they are anti-aligned. The energy difference between a state of aligned spins and one of anti-aligned spins is given by

$$E = -J \sum_{i,j} \sigma_i \sigma_j$$

(3)

where $\sigma_i$ (or $\sigma_j$) = +1 (or -1) and $J$ is the exchange energy between neighbors. A system with every spin parallel to each other spin is a ferromagnetic system. The alignment of the spins creates a macroscopic magnetization. The probability of finding the model in any particular state for the Ising Model is proportional to the Boltzmann factor:

$$P \propto e^{-E/T}$$

(4)

where $E$ is the energy defined above and $T$ is temperature. Evidently the number of possible spin configurations (microstates) is $2^N$, where $N$ is the total number of spins in the system. Below are two models representing the spins of a system.

![Figure 1](image.png)

Figure 1: The model on the left is representing spins by either shaded or not while the one on the right is using $\uparrow$ and $\downarrow$ for the spins.
In the Monte Carlo method, first a lattice structure is set up. The initial values for the individual spins and for the external field $H$ are defined. One element in the lattice is chosen, its spin changed from $+$ to $-$ or from $-$ to $+$ (flipped), and then the energy of the system with the new flipped spin is calculated. The element is left in its new position if the energy of the system is lowered by the flip (if the change in energy is negative). If the change in energy is positive, then the flip may or may not be retained. To decide whether to retain the flip we calculate a random number between 0 and 1. Next, we calculate, from the energy of the system, the probability (from the Boltzmann factor) that the system will attain that state. If that probability is greater than the random number, the spin flip is retained; if the probability is less than the random number, the spin is not flipped. The comparison to the random number allows the magnetic system to enter states of higher energy than that of a ferromagnet. These steps complete one Monte Carlo time step. Another element in the lattice is then chosen, and the process repeated for that element. This procedure is repeated a large number of times, so that each spin is given many chances to flip. The variables – energy, magnetization, susceptibility, heat capacity and absolute magnetization – are calculated, as well as the averages of those variables. The system’s dependence on temperature is carried by the temperature’s appearance in the probability term. We can visualize this process by the flow chart\(^8\) in the Appendix.

**Summary**

Both the Ising Model and the field of percolation are very rich and deep areas of study. The Ising Model is an extremely simple model of a magnetic system, which, despite its simplifications, captures the essential physics of magnetism. Of equal importance, the exploration of percolation has led to many new algorithms which are quite efficient, and can be applied to other problems as well. An investigation of percolation as well as the Ising Model, in their simplest forms, is a good way to see straightforward Monte Carlo methods in use, without being buried under several layers of complicated computer code. It is interesting to note that the Ising Model has even been used to model the activity of neural activity in the brain. From magnetization to percolation and even to neural activity, Monte Carlo simulations are a powerfully flexible method of describing our universe.

\(^8\)J.M. Yeomans “Statistical Mechanics of Phase Transitions”
Appendix

set up lattice sites \( i \)

define spins \( s_i \)

define \( \mathcal{H} \)

set counter, \( n = 1 \)

choose \( n_0, \tilde{n}_{\text{max}} \)

---

flip spin

\[ r = e^{-\Delta E/kT} \]

generate a random no. \( 0 < z < 1 \)

\( n = n + 1 \)

if \( r > z \)
accept flip

if \( r < z \)
reject flip

---

calculate variables, \( A_n \)

store for each step \( n > n_0 \)

---

calculate averages

\[ \langle E \rangle = \frac{1}{n_0} \sum_{n>n_0} A_n \]
References


