Monte Carlo, Metropolis and the Ising Model

Physics Computational Methods, Spring 2017

April 6, 2018

1 The Ising model

The Ising model is a simple, classical lattice model of a ferromagnet. In its simplest form, it is defined in terms of classical spins σ_j taking on the values ± 1 on a cubic lattice. The so-called reduced or dimensionless Hamiltonian of the Ising model can be written as

$$H/T = -\beta \sum_{jk} \sigma_j \sigma_k - h \sum_j \sigma_j$$

where J and h are dimensionless parameters. The first sum is over all nearest-neighbor pairs on the lattice, with each pair counted once, and the second sum is over all lattice sites. The parameter h represents the effects of an external magnetic field coupled to all spins, while the parameter J represents the coupling of each spin to its nearest neighbors. If $J > 0$, the coupling is ferromagnetic and in the limit $T \to 0$, i.e., $J \to \infty$, all spins are aligned. The partition function is given by

$$Z = \sum_{\{\sigma\}} e^{-H/T}$$

and the statistical average of any observable X is given by

$$\langle X \rangle = \frac{1}{Z} \sum_{\{\sigma\}} X \, e^{-H/T}.$$

The Ising model can be re-interpreted in many different ways; by defining an occupation number $n_x = (1 + \sigma_x)/2$, it can be used to model the liquid-gas transition.

When $h = 0$, the model is invariant under the global symmetry $\sigma_x \to -\sigma_x$. If this symmetry is unbroken, then we must have $\langle \sigma_x \rangle = 0$, while $\langle \sigma_x \rangle \neq 0$ implies *spontaneous symmetry breaking*. For dimension $d \geq 2$, the Ising model has a low-temperature (large J) phase with spontanously broken symmetry. The critical point where this behavior sets in is given by the critical coupling J_c. For the $d = 2$ Ising model on a square lattice, the critical coupling is known to be

$$\beta_c = \frac{\log\left(1 + \sqrt{2}\right)}{2} \approx 0.441$$

We can get some idea of what is going on using the Bragg-Williams approximation, which is a form of mean field theory. Suppose we have N total spins. For any configuration, there will be N_+ up spins and N_- down spins, with $N_+ + N_- = N$. The average magnetization of a spin in this configuration is $m = (N_+ - N_-)/N$ and m varies over $-1 \leq m \leq +1$. How many configurations will have the same value for m? This is

$$\frac{N!}{N_+!N_-!} = \frac{N!}{N_+!\,(N - N_+)!}.$$

These configurations will not all have the same energy, depending on the arrangements of spins, but we can estimate the energy as

$$H/T = -\beta d N m^2 - hNm$$

because dN is the number of bonds. We can solve for N_+ in term of m, finding $N_+ = N(1+m)/2$. The weight in Z of the configurations with a given m is then

$$\frac{N!}{N_+!\,(N - N_+)!} \exp\left[\beta d N m^2 + hNm\right].$$

The energy is smallest for $h = 0$ when $m = \pm 1$, but the entropic factor is maximized when $N_+ = N/2$ corresponding to $m = 0$. Using Stirling's approximation $n! \simeq n^n e^{-n}$, we find the weight to be given by

$$\exp\left\{N\left[\beta d m^2 + hm - \frac{1+m}{2}\log\left(\frac{1+m}{2}\right) - \frac{1-m}{2}\log\left(\frac{1-m}{2}\right)\right]\right\}$$

For $h = 0$, the entropy dominates when $d\beta < 1$ and the the most important configurations have $m = 0$, but for $\beta d > 1$, the most important configurations will have $m \neq 0$. Note that the importance of configurations off the peak falls off very rapidly due to the factor of N in the exponent. With a little work, you can expand the argument to the exponential around $m = 0$ to verify the estimate of the critical point. You can also show that the most important value of m satisfies

$$m = \tanh(2d\beta m + h)$$

which is the usual mean field equation.

2 Simulation and the Metropolis algorithm

Monte Carlo simulations work by creating a stream of lattice configurations that together approximate an ensemble with Boltzmann weighting. The probability of appearance of a given configuration a is proportional to its Boltzmann weight $\exp(-E^a/T)$. There are some similarities with the microcanonical ensemble, where time average is taken to equal ensemble average. However, Monte Carlo time, associated with the production of a stream of configurations, is not real time, and the "time" evolution is only loosely related to real time evolution.

Suppose we have an initial configuration $a = 0$, from which we will produce $a = 1, 2, ...$ sequentially, with each configuration determined by the previous one and some Monte Carlo algorithm. It is conceptually useful to consider an ensemble of initial configurations, each evolving to produce a stream of configurations. We can then associate with a given configuration a a probability $P(a)$ of appearing at a Monte Carlo time t. It is common to model the evolution of $P(a)$ by a master equation

$$\frac{dP(a)}{dt} = \sum_b \left[w(a \leftarrow b)P(b) - w\left(b \leftarrow a\right)P\left(a\right)\right]$$

where $w\left(b \leftarrow a\right)$ is the rate at which a configuration a becomes a configuration b. A time-independent equilibrium solution is given by *detailed balance*:

$$\frac{P\left(a\right)}{P\left(b\right)} = \frac{w(a \leftarrow b)}{w(b \leftarrow a)}$$

The ratio of the transition probabilities is given by Boltzmann weighting:

$$\frac{P\left(a\right)}{P\left(b\right)} = \frac{\exp\left(-E^a/T\right)}{\exp\left(-E^b/T\right)}$$

This only specifies the ratio of weights, not their value. For rapid equilibration and decorrelation (see below), we would like the w's to be as large as possible. This behavior is given by the *Metropolis algorithm*, for which some of transition probabilities are one. For the Metropolis algorithm, we define

$$w\left(b \leftarrow a\right) = \min\left(1, \exp\left(E^a/T - E^b/T\right)\right)$$

so that $w\left(b \leftarrow a\right) = 1$ if $E^b < E^a$, but is less than one if $E^b > E^a$. The essential algorithm is this: consider a change from a to b: if the change lowers the energy, always make the change. If it raises the energy, make the change with a probability given by $\exp\left(E^a/T - E^b/T\right)$. A simple algorithm is to compute $r = \exp\left(E^a/T - E^b/T\right)$; if r is greater than a random number distributed uniformly between 0 and 1, make the change. Of course, if $r > 1$, there is no need to bother generating a random number.

3 Code review

The code below is a basic implementation in python, without the bells and whistles you would find in production code. We begin by importing the numerical python library. Other libraries are needed for plotting. The routines *hot_ start* and *cold_ start* create a $ns \times ns$ array, with each element randomly ± 1 for a hot start and all $+1$ for a cold start. Many variables are global, including the linear lattice size *ns.*

```
import numpy as np
import matplotlib.pyplot as plt
import math

#------------------------------------------------------------------------#
#   Build the system
#------------------------------------------------------------------------#
def hot_start():
    lattice = np.random.random_integers(0,1,(ns,ns))
    lattice[lattice==0] =- 1

    return lattice

def cold_start():
    lattice = np.ones((ns,ns))

    return lattice
```

The presence of a boundary has a pronounced effect on spins near the boundary. To obtain a better approximation to a large bulk system, periodic boundary conditions are generally used. The routine *bc(i)* implements that boundary condition, so the system wraps around: $ns + 1 \rightarrow 0$ and $-1 \rightarrow ns - 1$.

```
#------------------------------------------------------------------------#
#   Periodic boundary conditions
#------------------------------------------------------------------------#
def bc(i):
    if i > ns-1:
        return 0
    if i < 0:
        return ns-1
    else:
        return i
```

The routine *mag* returns the magnetization for the current configuration, and can be modified to measure the energy. On the other hand, the *energy* routine returns only the part of the energy that depends on the spin at the (N, M) lattice site, and does not include the coupling β.

```
#------------------------------------------------------------------------#
#   Measure magnetization
#------------------------------------------------------------------------#
```

```
def mag(lattice):
    m = 0.
    for j in range(0,ns):
                for k in range(0,ns):
                    m += lattice[j,k]
    return m/(ns*ns)

#------------------------------------------------------------------------#
#   Calculate internal energy
#------------------------------------------------------------------------#
def energy(lattice, N, M):
    return -1 * lattice[N,M] * (lattice[bc(N-1), M] \
                               + lattice[bc(N+1), M] \
                               + lattice[N, bc(M-1)] \
                               + lattice[N, bc(M+1)])

def sum_nn(lattice, N, M):
    return (lattice[bc(N-1), M] + lattice[bc(N+1), M] + lattice[N, bc(M-1)] + lattice[N, bc(
```

The routines *update* and *sweep* are are where the actual Monte Carlo algorithm is located. In *update*, a random lattice site (j, k) is chosen for potential updating. This randomization is actually unnecessary, and *sweep* updates by moving sequentially through the lattice.

```
#------------------------------------------------------------------------#
#   The Main monte carlo loop
#------------------------------------------------------------------------#
def update(beta):
    #lattice = hot_start()

    for step in enumerate(range(ns*nw)):
        j = np.random.randint(0,ns)
        k = np.random.randint(0,ns)

        E = -2. * energy(lattice, N, M)

        if E <= 0.:
            lattice[j,k] *= -1
        elif np.exp(-beta*E) > np.random.rand():
            lattice[j,k] *= -1

def sweep(lattice, beta):
    acc = 0
    for j in range(0,ns):
            for k in range(0,ns):
                sum_nn = lattice[bc(j-1), k] + lattice[bc(j+1), k] + lattice[j, bc(k-1)] + l
                new_spin = -lattice[j,k]
                dE =-1*(new_spin-lattice[j,k])*sum_nn
                if dE <= 0.:
                    lattice[j,k] = new_spin
                    acc += 1
                elif np.exp(-beta*dE) > np.random.rand():
                    lattice[j,k] = new_spin
                    acc += 1
    accept = (1.*acc)/(ns*ns)
    #print("Acceptance: ",accept)
```

The main code initializes variables, does ninit initial sweeps to thermalize, followed by nsweeps sweeps, prints parameters and results, and also does a bit of plotting. While this is neither elegant nor efficient code, it is a common approach to simple simulations like this.

```
#----------------------------------------------------------------------#
#   Main
#----------------------------------------------------------------------#


ns = 25
ninit = 10
nsweeps = 250
beta = 0.4

print("Size = ", ns)
print("Initial sweeps = ",ninit)
print("Sweeps = ", nsweeps)
print("beta = ", beta)
accept = 0.0
lattice = cold_start()
for n in range(ninit):
    sweep(lattice, beta)
    #update(beta)
    m = mag(lattice)
    #print("Sweep: ",n, "Mag = " ,m)

m = mag(lattice)
#print("Mag = " ,m)
mav = 0
mlist = np.ones(nsweeps)

for n in range(nsweeps):
    sweep(lattice, beta)
    #update(beta)
    m = mag(lattice)
    mav += m
    mlist[n]=m
    #print("Sweep: ",n, "Mag = " ,m)

mav = mav / nsweeps
print("Average m:", mav)
plt.plot(mlist)
plt.show()
```

4 What can we measure?

When $h = 0$, the $\sigma \to -\sigma$ symmetry implies $\langle \sigma \rangle = 0$, unless the symmetry is spontaneously broken. The average magnetization for a configuration $a$is given

$$m^a = \frac{1}{N_s} \sum_j \sigma_j^a$$

The average magnetization for a set of configurations C of size N_c is

$$m^C = \frac{1}{N_C} \sum_{a \in C} m^a$$

In the limit where N_C goes to infinity with some given parameters, we suppose that $m^c \to \langle m \rangle$, the ensemble average over all configurations. Our expectation is that $m^c \neq 0$ for $J > J_c$. The magnetic susceptibility can be obtained as

$$\chi^C = \frac{1}{N_C} \sum_{a \in C} \left(m^a - m^C \right)^2$$

At a second order phase transition, this is expected to be singular at J_c. We can also measure the energy. Usually we measure something related to the energy, for example

$$e^a = \frac{1}{dN_s} \sum_{jk} \sigma_j^a \sigma_k^a$$

$$e^C = \frac{1}{N_c} \sum_{a \in C} e^a$$

The specific heat is

$$C_V = \frac{1}{N_C} \sum_{a \in C} \left(e^a - e^C \right)^2$$

The specific heat also becomes singular at the critical point.

Problem: As a warm-up, graph m as a function of β. Use as many value of β above and below the transition to get a feeling for how m behaves. Do this for lattices of different sizes. Try 5×5, 10×10, 20×20, however large you think is reasonable. Explain how and why you set the various initialization variables to the values you chose. What do you observe in the high temperature phase? What do you observe in the low-temperature phase? How well can you estimate the known critical coupling β_c from the simulation data?

5 What are the pitfalls?

One pitfall which we do not discuss in detail is the failure of ergodicity, *i.e.*, a Monte Carlo algorithm may not explore all of phase space. There are some plausible algorithms that fall into repetitive cycles and don't explore all configurations. Another subtle problem is an inadequate random number generator, one that is not "random" enough, or has too short a cycle. Other, more physical problems are discussed below.

5.1 time to equilibrate

We have argued that an equilibrium ensemble of the Metropolis algorithm receives correct Boltzmann weighting. However, we always start from a particular configuration. Two common choices for a starting configuration are a *cold start*, in which all spins are aligned, and a *hot start*, where all spins are randomly chosen to be up and down. There are good choices for $T = 0$ and $T \to \infty$, respectively, but are not not representative for temperatures between these extremes. It is easy to see that we must wait some time for non-equilibrium behavior due to initial conditions to die away. Those initial configurations are monitored to assess equilibration, but not used for equilibrium measurements. A simple, minimal criteria is that one must wait until both a hot start and a cold start display similar behavior

Problem: Study the equilibration time by plotting m^a, the average of spins in a configuration a, as a function of simulation time for hot and cold starts, using at least three values of β above and three widely spaced values below the critical coupling. Does the equilibration time appear to change as the size of the lattice changes? If so, how?

5.2 finite volume effects

Spontaneous symmetry breaking cannot truly occur in any system with a finite number of degrees of freedom. Even at very low temperatures, there is some small probability that the spins will flip sign. This is often referred to as a tunneling event.

Problem: By plotting m^a, the average of spins in a configuration a, as a function of Monte Carlo time, observe tunneling events for $J > J_c$ on lattice of at least two different sizes at the same value of J. Does the time between tunneling events change as the lattice size changes? If so, how?

Problem: Tunneling implies that the long-term average of m will always be zero. One common alternative is to measure instead $|m|$. Add code to measure $|m|$ and make a plot of J versus $|m|$ for lattice of at least two different sizes. Do you see any problems with using $|m|$ to locate the critical point?

5.3 autocorrelation time

The autocorrelation time *for a given variable* here the magnetization) can be defined loosely as

$$A(t) = \frac{1}{N_c} \sum_a (m_a - \langle m \rangle)(m_{a+t} - \langle m \rangle)$$

but in practice one must be careful that the sum over a stops before $a+t$ moves past the last configuration. Typically $A(t)$ will decay roughly as $\exp(-t/\tau)$, where $\tau$is the autocorrelation time. A large value of τindicates a significant degree of correlation between successive configurations. In other words, they are not statistically independent. The precise value of τ will depend on the parameters and on the observable measured. Generally speaking, at high T (small J), the correlation time is small. At very low temperatures, it can be large. The autocorrelation time also becomes large near the critical point. This is known as *critical slowing down* and is also seen in real physical systems. It can be made physically plausible by remembering that C and χ diverge at the critical point, indicating that large statisitical fluctuations are common in the critical region. Critical slowing down means that runs near a critical need to be much longer than those away from the critical point to have good control of errors.

Two steps are commonly taken to control the underestimation of statistical error due to autocorrelation. The first is that not all configurations are measured. As discussed above, some initial set of configurations are discarded as the system equilibrates. It is common to only measure one configuration out of every τ or so, so that successive measured configurations are less correlated. If τ is 5, then measuring every 15'th configuration will significantly reduce autocorrelation. Reducing sampling to the point where no autocorrelation is detectable is very inefficient. In practice, it is thus necessary to correct for autocorrelation in statistical analysis of the simulation data. This typically produces larger statistical errors than one would naively estimate.

Problem: As a final problem, calculate the average value of C_V for lattices of various sizes for several values of J above and below the critical coupling. Graph the behavior and interpret what you see. Explain what problems you see in obtaining an approximation to the true equibrium behavior of an infinite lattice. Explain as quantitatively as you can how you would overcome these problems.