Week 7

Markov Chain Monte Carlo

1 A game of chance

Suppose your friend Helena offers to play the following game of chance with you. She is going to flip a coin, and if the coin comes up heads, then you pay her \$1, but if the coin comes up tails, then she pays you \$1. You like gambling, so you agree to play with her, and in fact your find the game so fun that you agree to play the game many many times. In other words, Helena will flip the coin a large number of times in a row, and each time you either pay her or she pays you depending on whether or not the coin comes up heads or tails. Little did you know, however, that Helena has weighted the coin so that it's 1.1 times as likely to come up heads (and win her a dollar) than it is to come up tails (and win you a dollar). The weighting of the coin is small enough that you won't really be able to tell the difference when you start playing the game, but Helena knows that if you play for a large number of flips, she will make a lot of money.

> *How much money should Helena expect to make on average per flip if she plays this game a large number of times?*

If you know a bit about probabilities, you can calculate the answer to this question with a little calculation by hand on a piece of paper. We'll do that calculation below. Another alternative would be to simply play the real game a large number of times, count how much money Helena makes, and then divide this by the number of times the game was played. Yet another approach, the one we'll illustrate here, is to simulate the game on computer and see how much money Helena makes on average per flip. Before we get into how one might simulate this, let's take a look at the results of some simulations of this game. Our notation is that n is the total number of flips, n_H is the number of times the coin comes up heads, n_T is the number of times the coin comes up tails, and E is Helena's average earnings per flip and is computed as

$$E = \frac{n_H(\$1.00) + n_T(-\$1.00)}{n} \tag{1}$$

Simulation results are as follows with E being rounded to the nearest tenth of a cent in each case.

n	n_H	n_T	E
10	8	2	\$0.600
100	52	48	\$0.040
1000	501	499	\$0.002
10000	5217	4783	\$0.043
100000	52669	47331	\$0.053
1000000	524396	475604	\$0.049
10000000	5236550	4763450	\$0.047

Do these simulation results make sense? Well let's do the analytical calculation. Since the probability of heads is 1.1 times the probability of tails we have

$$p_H = 1.1p_T \tag{2}$$

On the other hand, the probabilities must sume to 1, so we have

$$p_H + p_T = 1. \tag{3}$$

This is a system of two linear equations in two unknowns p_H and p_T with a unique solution. We can solve this system by plugging $1.1p_T$ for p_H in the second equation to obtain

$$1.1p_T + p_T = 1 \tag{4}$$

which implies

$$p_T = \frac{1}{2.1} \approx 0.476, \qquad p_H = 1 - p_T \approx 0.524. \tag{5}$$

Helena's average earnings per flip will then be the probability of heads times her earnings per heads flip plus the probability of tails times her earnings per tails flip:

$$E \approx 0.524(\$1.00) + 0.476(-\$1) = \$0.048. \tag{6}$$

It does indeed seem that for large numbers of flips, the simulation has produced the correct earnings per flip! So how exactly did the simulation work? Here were the basic steps:

1. Initialize the coin in whatever state you choose, either H or T.
2. Flip a *fair* coin (it's gotta be a different coin since Helena's coin is rigged) to propose moving to a new state H or T.
3. If the proposed state is the same as the current state, accept the move (which does nothing).
4. If the proposed state is different than the current state, then move to the new state with "acceptance" probability

   $$p_{\text{accept}} = \min\left(1, \frac{p_{\text{proposed}}}{p_{\text{current}}}\right) \tag{7}$$

   In practice, this can be done by choosing a random number r from the uniform distribution on the interval $[0, 1]$ and accepting the state if $r < p_{\text{accept}}$. Can you see why this makes sense?
5. Repeat this over and over to generate a sequence of H and T. The longer your run the program, the more the ratio of the number of heads to the number of tails n_H/n_T will tend to the ratio p_H/p_T.

This algorithm is the so-called **Metropolis Algorithm**. Notice that the algorithm only requires knowing the ratio of the probability of the proposed state to the current state. In general, the metropolis algorithm and its generalizations allow one to do the following:

> *Given a system with a finite number of states* $1, 2, \ldots, N$ *and associated probabilities* $p_1, p_2, \ldots, p_N$ *such that only the ratios of probabilties* p_i/p_j *are known, generate a sequence of states* $s_1, s_2, \ldots, s_M$ *such that if* n_i *is the number of times state* i *appears in the sequence, then*
>
> $$\frac{n_i}{n_j} \approx \frac{p_i}{p_j}. \tag{8}$$

In other words, the Metropolis algorithm generates a sequence of states where each state approximately appears a number of times that is proportional to its probability. As the number of metropolis steps increases, the equation tends to be closer to true. The key point is that only the *relative* probabilities of the states need to be known in order to run the algorithm; the overall normalization that makes the probabilities sum to 1 does not need to be known.

2 Thermodynamics of solids - a motivating example

A solid is made of macroscopic numbers 10^{23} of atoms, and the possible states of such solid are even greater in number! Imagine for simplicity that each atom can have spin up or down state (we are ignoring the position or momentum degree of freedom of the atom), and just look at the spin degrees of freedom, and you will get $2^{10^{23}}$ possible microscopic magnetic states – an absolutely enormous number. This means that there is no hope of, for example, storing all of the possible states of a solid in computer memory.

Now you may remember from a statistical mechanics class that all of the thermodynamic properties of a solid can be derive from its partition function Z and that the partition function is a sum over states of the Boltzmann factor for each state:

$$Z = \sum_s e^{-E_s/kT}, \tag{9}$$

where E_s is the energy of state s, k is Boltzmann's constant, and T is Kelvin temperature. If the number of states is as large as it is for a real solid, it becomes impossible to compute this sum on computer, but that's ok! There's another way. The probability that the system will ocuppy a given state in thermal equilibrium is

$$p_s = \frac{e^{-E_s/kT}}{Z} \tag{10}$$

It follows that the relative probabilities of states is the ratio of the associated Boltzmann factors

$$\frac{p_s}{p_\ell} = \frac{e^{-E_s/kT}}{e^{-E_\ell/kT}} \tag{11}$$

Notice that Z has gone away! The ratio of the probabilities of any two states is easy to compute. Moreover, usually what we're after in thermodynamics is the average value of some observable O whose value in thermal equilibrium will simply be its average

$$\langle O\rangle = \sum_{s}^{N} p_s o_s. \tag{12}$$

where o_s is the value of the observable when the system is in state s. By running the Metropolis algorithm to generate a sequence of states $s_1, s_2, \ldots, s_M$ that occur in proportion to their probabilities, we can approximate the above ensemble average as follows:

$$\langle O\rangle \approx \sum_{i=1}^{M} \frac{n_i}{M} o_i. \tag{13}$$

where n_i is the number of times state i appeared in the sequence.

3 Metropolis-Hastings Algorithm

The Metropolis algorithm applied above is a special case of a more general class of algorithms called Markov Chain Monte Carlo algorithms, or MCMC for short. A Markov chain is a stochastic process the generates a sequence of possible states in which the probability of each state depends only on the previous state. There are a limitless number of Markov chains one can imagine, but MCMC works by generating Markov chains where the number of times a given state tends to appear is proportional to its probability. There are many different MCMC algorithms, but here we describe a generalization of the Metropolis algorithm called Metropolis-Hastings.

Consider a system with a finite number of states $1, 2, \ldots, N$ with associated probabilities $p_1, p_2, \ldots, p_N$. This means that

$$\sum_{n=1}^{N} p_n = 1, \qquad p_n \geq 0. \tag{14}$$

An observable O of such a system is a physical quantity that has a certain value o_n if the system occupies state n. The average value of an observable is defined as

$$\langle O\rangle = \sum_{n=1}^{N} p_n o_n. \tag{15}$$

For example, for a system in equilibrium with a heat bath, the the states could be states of definite energy, the probabilities would be the probability of finding the system in one of these states, and the energy itself would be an observable – each state has a certain associated energy value.

It can often be difficult to compute averages of observables for physical systems for various reasons. For example, the system might have an enormous number of states causing naive numerical evaluation of averages to be difficult or impossible. In such cases, a common workaround is to generate a representative random sample $s_1, s_2, \ldots, s_M$ of states and compute the sample average

$$\langle O \rangle_M = \frac{1}{M} \sum_{n=1}^{N} o_{s_n} \tag{16}$$

As M becomes large, the sample average will converge to the average provided the samples are chosen appropriately;

$$\langle O \rangle_M \to \langle O \rangle, \qquad M \to \infty. \tag{17}$$

n a computational setting, the Markov chain can be generated according to some algorithm. Perhaps the most well-known, simplest, and most elegant of all of these is the so-called **Metropolis-Hastings algorithm**:

1. Initiate the sequence of samples in a state s_1 of your choosing, and successively generate states in the following way:

2. If s is the current state, propose a new state s' according a conditional probability distribution q, namely $q(s'|s)$ is the probability of proposing s' given the current state is s.

3. Accept the proposed state, namely add it as the next state in the sample sequence, with probability

$$A(s'|s) = \min\left(1, \frac{p_{s'}}{p_s}\frac{q(s|s')}{q(s'|s)}\right) \tag{18}$$

   If the state is not accepted, in other words if it is rejected, add the current state s to your sequence instead as the next state.

4. Repeat these steps for a large number of samples.

The proposal distribution is often chosen to be symmetric

$$q(s|s') = q(s'|s). \tag{19}$$

Choosing this sort of proposal distribution simplifies the acceptance probability to

$$A(s'|s) = \min\left(1, \frac{p_{s'}}{p_s}\right), \tag{20}$$

and in this case the algorithm is often called simply the **Metropolis algorithm**. Proving that the Metropolis Hastings and Metropolis Algorithms work properly is beyond the scope of the course, but we encourage the reader to study the proof in the literature on MCMC if desired.