Worksheet week 8 – Monte Carlo method for the 1D Ising model

1 Purpose

1. Write a C++ code to implement Markov chain Monte Carlo using the Metropolis algorithm for the one-dimensional Ising model.
2. Use your code to explore the behaviour of the system for various temperatures, applied field strengths, and interaction strengths.
3. Compare your code to analytical results to ensure that it's working correctly.

2 Important concepts

Importance sampling:
Most systems of interest contain incredibly large numbers of particles and each particle may have an incalculable number of states — so in general we can not even hope to directly evaluate partition functions Z. For any realistic model the number of particles is of order 10^{23}. We can, however, use *importance sampling* to generate likely configurations over which to sum. The idea is to somehow focus our efforts on computing the configurations of spins (microstates) that are more likely (and thus contribute more to the averages).

We know that, in equilibrium, our probability distribution function should be:

$$p_m = \frac{e^{-E_m/k_BT}}{\sum_m e^{-E_m/k_BT}} = \frac{1}{Z} e^{-E_m/k_BT}, \tag{1}$$

where m denotes a particular microstate.

If we assume that we are able to generate microstates from this probability distribution, then we can substitute it into the expression for calculating the average of a quantity O:

$$\langle O \rangle \approx \frac{\sum_m^M \left(O_m e^{-E_m/k_BT}/p_m \right)}{\sum_m^M \left(e^{-E_m/k_BT}/p_m \right)} = \frac{1}{M} \sum_m^M O_m. \tag{2}$$

We see that, by substituting Eq. (1), the Boltzmann factors and partition functions have cancelled out, and the expectation value is simply the average over the observables O_m measured from each microstate generated. The summation here is over M sample states generated according to the probability distribution function p_m. The approximation becomes exact in the limit $M \to \infty$. The problem still remains of how to generate sample states according to the probability distribution function p_m. This is a serious issue, since we still cannot directly evaluate the denominator of Eq. (1).

Markov processes:
The answer to this quandary is a powerful method called Markov chain Monte Carlo. It utilises Markov chains to generate a suitable probability density for importance sampling without the need to directly calculate normalised probability functions.

Suppose that we have some stochastic process generating quantities $\{X_0, X_1, \cdots\}$ at times $t_0, t_1, \cdots$ respectively (with $t_0 \leq t_1 \leq \cdots$). Then we can define $P(X_n | X_0, X_1, \cdots, X_{n-1})$ as the conditional probability that given the prior sequence $\{X_0, X_1, \cdots, X_{n-1}\}$, the next quantity generated will be X_n (at time t_n). The relationship between this conditional probability and the joint probability $P(X_0, X_1, \cdots, X_{n-1}, X_n)$ that the sequence $\{X_0, X_1, \cdots, X_n\}$ is generated, is

$$P(X_0, X_1, \cdots, X_{n-1}, X_n) = P(X_n | X_0, X_1, \cdots, X_{n-1}) P(X_0, X_1, \cdots, X_{n-1}). \tag{3}$$

A stochastic process has the Markov property if

$$P(X_n|X_0, X_1, \cdots, X_{n-1}) = P(X_n|X_{n-1}). \tag{4}$$

This means that a Markov process has future probabilities which are determined solely by its most recent value. A Markov chain is a sequence $\{X_0, X_1, \cdots\}$ that is generated by a stochastic process that has the Markov property and obeys Eq. (4). $P(X_n|X_{n-1})$ is called the transition kernel.

Balance and detailed balance:
Balance is one the two conditions that must be satisfied for the Markov chain Monte Carlo method to produce the correct equilibrium statistics (as mentioned in the lecture, the other condition is *ergodicity*). To demonstrate what this condition means, consider a population of random walkers on a lattice (in an arbitrary geometry/dimensionality), where the walkers move independently of each other. For each step in the random walk, the walkers choose to randomly jump to one of their nearest-neighbour lattice sites.

After some number of steps, the density of walkers at a given point m on the lattice is denoted by ρ_m. At the next step in the random walk, the net change in the density of walkers at the point m will be the sum of all walkers arriving at m, minus the sum of all walkers leaving m, i.e.:

$$\begin{aligned}\Delta\rho_m &= \sum_n \left[\Delta\rho_{n\to m} - \Delta\rho_{m\to n}\right] \\ &= \sum_n \left[\rho_n P(m|n) - \rho_m P(n|m)\right]\end{aligned} \tag{5}$$

where $P(n|m)$ is the conditional probability that a walker at m it makes a transition to n. After a sufficiently long time, we would expect the density of walkers at a given lattice site to reach an equilibrium condition, i.e. $\Delta\rho_m = 0$ for all m (on average). This is the condition of balance.

Detailed balance goes a step further by enforcing not just that the sum in Eq. (5) vanishes, but that *each individual term* in the sum also vanishes, i.e. $\Delta\rho_{m\to n} = \Delta\rho_{n\to m}$. This leads to the condition:

$$\frac{\rho_m}{\rho_n} = \frac{P(m|n)}{P(n|m)}. \tag{6}$$

For the random walk example, this condition means that the mean rate at which walkers move between any two lattice sites m and n is equal and opposite, and hence there is no net movement on average.

Metropolis Algorithm:
Here we will consider the Metropolis algorithm, which can be used to efficiently generate random variates distributed over a high-dimensional probability distribution. With this method we will be able to evaluate complex ensemble averages like Eq. (2), which otherwise we would have little hope of evaluating.

In the Metropolis algorithm, the conditional probability $P(n|m)$ (i.e. the probability of making a transition to state n, given that the system is currently in state m) takes the form:

$$P(n|m) = \min\left(1, \frac{p_n}{p_m}\right) \tag{7}$$

where p is some probability distribution that we are interested in generating states from. For the Ising model, we will use the probability distribution in Eq. (1), which gives

$$P(n|m) = \min\left(1, \exp\left(-\frac{\Delta\mathcal{E}_{m\to n}}{k_B T}\right)\right), \tag{8}$$

where $\Delta\mathcal{E}_{m\to n} = \mathcal{E}_n - \mathcal{E}_m$ is the change in energy.

To implement the Metropolis algorithm, the system is initiated in an arbitrary configuration (at least in principle—in practice, there are often particularly good or bad ways to initiate the system). In our case, we will be dealing with configurations of spins, $\sigma = \pm 1$, so you could choose these to be e.g. all up, all down, or randomised. Once the system is initiated, the following steps are repeated a large number of times:

1. Choose a lattice site (either sequentially or at random), and propose to flip the spin at that site. (There are also more advanced algorithms that flip multiple spins, but we'll just focus on single spin flips here.)
2. Calculate the change in energy $\Delta\mathcal{E}$ that results from the proposed flip.
   (a) If $\Delta\mathcal{E} \leq 0$, accept the proposed flip.
   (b) If $\Delta\mathcal{E} > 0$, generate a uniform random number r, such that $0 \leq r < 1$. Let $\alpha = \exp\left(-\Delta\mathcal{E}/k_B T\right)$.
      i. If $r \leq \alpha$, accept the proposed flip.
      ii. If $r > \alpha$, reject the proposed flip.

There are a few things to consider here:

- You generally want to repeat the process $\gg N_\mathrm{s}$ times, where N_s is the number of spins in the lattice. A good starting point would be $\sim 100 N_\mathrm{s}$ proposed flips (i.e. 100 'sweeps' of the full lattice), but $\sim 1000 N_\mathrm{s}$ or more should also be possible. Remember that the precision of your averaged observables will scale as $\mathcal{N}^{-1/2}$, where $\mathcal{N}$ is the number of samples.
- A good choice of initial state is usually a highly probable state at the chosen temperature (if you know how to generate one). The further your initial state is from the most likely state, the longer the system will take to thermalise.
- For the Ising model, you generally only save your observables once per sweep of the lattice. If you save more frequently than this, your states will be highly correlated with one another, leading to oversampling of particular regions of the phase space.
- You will need to include a period of 'burn-in' at the beginning of the evolution, during which time you don't save any data. Burn-in accounts for the fact that the initial state is generally not 'typical' at the chosen temperature—it is usually chosen arbitarily, rather than on physical grounds. You therefore want to erase all 'memory' of where you started the Markov chain (more precisely, you want to wait until your states are sufficiently decorrelated from the initial state).
- If we were to try and make too drastic a change each time we do step 1 (e.g. by flipping multiple spins at once), we would create a large energy change $\Delta\mathcal{E}$ for many of our proposed flips, and we would therefore end up with a high probability of rejection (due to the exponential dropoff of acceptance for $\Delta\mathcal{E} > 0$). This would reduce the efficiency of the algorithm, and could lead to the system getting stuck without reaching equilibrium. We will therefore stick to single spin flips each step.

Choice of boundary conditions:
Generally, analytical predictions of thermodynamic behaviour describe the system in the *thermodynamic limit*, which means that the system extends to infinity in all directions. But of course, numerical simulations can only ever be performed in finite domains, and hence they will always suffer from *finite-size effects*. Essentially, what this means is that the observables measured from the simulations will not match the theoretical predictions, but will instead only *approach* the predictions as the simulation system size $L \to \infty$.

One important aspect that contributes to finite-size effects is what happens at the boundary of the system; hence the choice of boundary conditions is important. The best way to minimise finite-size effects is to use periodic boundary conditions, because in doing so you maximise the lengthscale over which the system "looks" like the thermodynamic limit (because you can start at any grid point and travel a distance of at least $L/2$ in any direction before you "wrap around" and start getting closer to the starting point again). As such, we will adopt periodic boundary conditions for all of our Ising model simulations. Note that this only affects the interaction term.

3 Monte Carlo method for the 1D Ising model

In this worksheet, we will consider the one-dimensional Ising model, which has Hamiltonian:

$$H = -J \sum_{\langle i,j \rangle} \sigma_i \sigma_j - \mu B \sum_{i}^{N_\mathrm{s}} \sigma_i. \tag{9}$$

Here, $\sigma_i = \pm 1$ is the spin at site i, J is the spin–spin interaction strength, $\mu = 1$ is the magnetic moment of each spin, and B is the applied magnetic field strength. The notation $\langle i, j \rangle$ denotes a sum over pairs of nearest neighbours, i.e. the sum includes interactions between spins 0–1, 1–2, 2–3, 3–4, $\cdots$ [with periodic boundary conditions, the final term is $(N_\mathrm{s} - 1)$–0].

The interesting observables for the Ising model are the energy per spin $E = \mathcal{E}/N_\mathrm{s}$, and the magnetisation per spin:

$$M = \frac{\mathcal{M}}{N_\mathrm{s}} = \frac{1}{N_\mathrm{s}} \sum_i \sigma_i. \tag{10}$$

From here on, I'll use curly symbols $\mathcal{E}$ and $\mathcal{M}$ for the unnormalised observables, and regular E and M for the normalised versions.

3.1 Exercise – detailed balance:

Show (using pen and paper) that the Metropolis update rule, Eq. (7), satisfies the detailed balance condition (6).

3.2 Exercise – non-interacting 1D Ising model:

Setup:
We will start with the case $J = 0$, $B = 1$. Write a C++ script that implements the Metropolis algorithm for a 1D spin chain (see description above) with an arbitrary number of spins N_s and temperature T. Start with something like $N_\mathrm{s} = 20$, $T = 1$.

If you completed the final exercise in last week's worksheet, you should be able to build on the code you wrote there, and just add the Metropolis condition for accepting/rejecting spin flips. It is recommended that you store the spin configuration in a matrix using the matrix class that was provided in last week's workshop (this will make it easy to generalise your code to two dimensions for the assignment!). You might also like to occasionally (or just at the end of the simulation) output the current spin configuration to the terminal, e.g. in a form like:

```
O - O O O - - O - O O - - O
```

where "O" corresponds to $\sigma = +1$, and "-" corresponds to $\sigma = -1$.

First run:
Start by running the code for $\sim 10 N_\mathrm{s}$ steps (where one step is one proposed spin flip). Every step, save the energy and magnetisation measurements to a file, and plot them as a function of the iteration number (don't include a burn-in time yet).

What do you see? Is there a sudden change in the energy at early times? If not, try increasing N_s. You should be able to see regions where the curves are flat and nothing changes—this is where the Metropolis algorithm has rejected a number of flips consecutively. Try lowering the temperature towards zero and see how the curve changes. (For comparison, you might want to look at the data I presented in the lecture notes.)

Minimising temporal correlations:
Try running a bit longer, e.g. for $\sim 100N_s$ steps. But now only save the output once per *sweep* of the full lattice (i.e. every N_s steps). Can you see the difference in the curve? The fluctuations should look a lot less continuous now (as a function sweep number).

Remember that what you're trying to achieve with the Markov chain is, ideally, a *random sample* of states from your probability distribution p_m. So ideally your subsequent samples should be completely independent of one another. By sampling once per sweep, the goal is to reduce the effects of correlations between our samples, so that they 'look' random. It is possible (but not particularly straightforward) to work out the *correlation time*, which estimates how many Markov chain steps it takes for the 'memory' of a given state to drop to essentially zero. A good indicator of the correlation time is the rapid period of evolution at the beginning of your simulation. In practice, the correlation time is longer than a full sweep of the lattice, but sampling once per sweep reduces the temporal correlations to an acceptable level.

Looking at your data, decide on an appropriate burn-in time for the beginning of the simulation. You will probably see that the observables take some time to settle to their equilibrium values. Estimate this time, and then add in a 'safety factor' of 10 or more, just to be sure. So if the curves seem to settle after 10 lattice sweeps, choose a burn-in time of 100 lattice sweeps.

Calculating averages:
Now that you've set up your Monte Carlo code, and you've minimised the temporal correlations, you can start taking data. Have a look at the mean energy per spin and magnetisation per spin, $\langle E \rangle$, $\langle M \rangle$. How do these values vary as you change T at fixed B? Is this what you would expect to happen? What about if you fix $T = 1$ and vary B (note that B can be both positive and negative)?

Important: Note that, for maximum efficiency, you should calculate the total energy and total magnetisation at the beginning of your loop, but only calculate $\Delta\mathcal{M}$ and $\Delta\mathcal{E}$ at each step (you need to calculate $\Delta\mathcal{E}$ for the Metropolis acceptance rule anyway). Then, if the proposed spin flip is accepted, add $\Delta\mathcal{M}$ and $\Delta\mathcal{E}$ to running totals. Based on Eqs. (9) and (10), you should be able to write two simple functions to calculate the change in energy and magnetisation if a single spin is flipped. For example, for the total magnetisation, you know that flipping $+1 \to -1$ will give $\Delta\mathcal{M} = -2$, while flipping $-1 \to +1$ will give $\Delta\mathcal{M} = +2$. You can therefore see that $\Delta\mathcal{M} = -2\sigma_{\text{initial}}$, where σ_{initial} is the spin value before flipping. You can do something similar for the energy.

Benchmarking:
To check that your simulation is working, you can check the magnetisation per spin against the exact solution for the 1D Ising model:

$$M(B,T) = \frac{e^{\beta J}\sinh(\beta B)}{\left(e^{2\beta J}\sinh^2(\beta B) + e^{-2\beta J}\right)^{1/2}}, \tag{11}$$

where $\beta = 1/(k_B T)$ is the inverse temperature.

Set up your code to do a loop over ~ 10 field strengths at a fixed temperature $T = 1$ (e.g. in the range $-2 \leq B \leq 2$). Save the mean energy and magnetisation per spin at each field strength, and plot these as a function of B. Compare the magnetisation to the theoretical result to see if you get agreement.

3.3 Exercise – interacting 1D Ising model:

Setup:
Now add the interaction term to the Hamiltonian in your code. Make sure that your sum doesn't double count interactions! To implement periodic boundary conditions you need to ensure that, for the first spin on the lattice, $i = 0$, the left neighbour must be $i - 1 = N_\mathrm{s} - 1$. Likewise, for the final spin $i = N_\mathrm{s} - 1$, the right neighbour must be $i + 1 = 0$. You can make use of the % (modulo) operator in C++ to always be sure you get the correct indices for the neighbouring spins, even at the boundary. C++ treats the modulo of negative numbers a little weirdly, but you can get the left and right neighbouring indices using:

```
int iPrev = (i + N - 1) % N;
int iNext = (i + 1) % N;
```

Important: For maximum efficiency, you again want to only calculate $\Delta\mathcal{M}$ and $\Delta\mathcal{E}$ each step, and add these to a running total. Because the interaction term in the Hamiltonian only depends on the nearest-neighbours, flipping a spin i will only affect the energy contributions from the interactions $(i-1) \leftrightarrow i$ and $i \leftrightarrow (i+1)$. You can therefore calculate $\Delta\mathcal{E}$ by just looking at how these two interaction terms change. Specifically, think about how the energy $\mathcal{E}_i = -J(\sigma_i\sigma_{i-1} + \sigma_i\sigma_{i+1})$ changes if σ_i is flipped.

Benchmarking:
Once you've added in the interaction and managed to get the boundary conditions working, set $J = 0.5$ and $T = 1$, and perform the same loop over magnetic fields as in the previous section. Make sure that your magnetisation still agrees with the analytical prediction, Eq. (11). What do you notice in comparison to the $J = 0$ case? What happens in the limit $J > T$? Why?

Larger system sizes:
Now that you've got everything working, what's the largest system you can go to? Try $N_\mathrm{s} = 64$, $N_\mathrm{s} = 128$, or more! But make sure you're not saving too much data to your output files (this shouldn't be an issue if you're saving once per sweep). If it's getting too slow, you could try to reduce the number of sweeps of the lattice.

A note on system sizes: often in computational physics, you see numerical grid sizes that are chosen in powers of two, i.e. 2^n. This is often a convenient choice because computers naturally store information in powers of two (since everything is made of bits). For the purposes of this course, it will be particularly useful to use grid sizes in powers of two when we use OpenMP for parallelisation, because this will allow for the most even distribution of the workload across threads (which are themselves also chosen in powers of two).