Lauren E. Hayward Sierens

Tutorial 8: Classical Monte Carlo Simulation of the Classical 2D Ising Model

October 26, 2017

In this tutorial, we will study phase transitions within the classical two-dimensional Ising model, with Hamiltonian

$$H = -J \sum_{\langle ij \rangle} \sigma_i \sigma_j - h \sum_i \sigma_i,$$

where $\sigma_i = \pm 1$, J is the coupling strength, h is a magnetic field and $\sum_{\langle ij \rangle}$ denotes a sum over nearest neighbours. We will consider simulations on a lattice with periodic boundary conditions and set $h = 0$. In the thermodynamic limit, the critical temperature is known to be

$$\frac{T_c}{J} = \frac{2}{\log(1+\sqrt{2})} \approx 2.269.$$

We will use and modify the two Python programs `ising2d_mc.py` and `plot_ising.py` throughout this tutorial in order to implement Monte Carlo (MC) methods that estimate T_c and compare with this known exact solution.

1 Monte Carlo algorithm

Consider the Monte Carlo program `ising2d_mc.py`, which is designed to perform a Monte Carlo simulation (using the single-spin-flip Metropolis algorithm) and record measurements of the system's energy E and magnetization M.

a) Examine the section of the code that computes the two-dimensional `neighbours` array, which is used when calculating the system's energy. The code is already written such that `neighbours[i,0]` and `neighbours[i,1]` store the lattice location of spin `i`'s rightward and upward neighbours, respectively. Modify the code such that it will also store spin `i`'s leftward neighbour in `neighbours[i,2]` and its downward neighbour in `neighbours[i,3]`.

Hint: Don't forget to consider the periodic boundary conditions.

b) Examine the `sweep()` function, which proposes a chain of `N_spins` Monte Carlo updates. Compare with the algorithm given during the lectures and convince yourself that this code is implementing the single-spin-flip Metropolis algorithm.

c) Implement a more efficient way of calculating the energy difference `deltaE` within the `sweep()` function. The given implementation calculates this energy difference by using the `getEnergy()` function, which involves iterating a loop `N_spins` times. However, you should be able to calculate `deltaE` by summing only four terms.

Hint: To appreciate the difference in time required for these two different implementations, set `animate = False` and increase the linear size `L` when you run the two versions of the code.

d) Run the code for various values of `L`, `J`, `n_eqSweeps` and `n_measSweeps` and explain how each of these parameters affects the resulting animated samples.

e) Run your code with `L=4`, `T_list = np.linspace(5.0,0.5,10)`, `n_eqSweeps=1000` and `n_measSweeps=10000`. The code will generate files in a directory called `Data` that will store the energy and magnetization corresponding to each of your sampled system configurations. (In Question 2, we will analyze and plot the resulting data.)

Hint: Set `animate = False` for this part so that the code runs faster.

2 Estimating the critical temperature

Recall from lecture that the specific heat C_V and susceptibility χ can be expressed as

$$C_V = \frac{\langle E^2\rangle - \langle E\rangle^2}{T^2}, \qquad \chi = \frac{\langle M^2\rangle - \langle M\rangle^2}{T},$$

where E is the energy and $M = \sum_i \sigma_i$ is the magnetization. For our Monte Carlo calculations on finite lattices, there is no spontaneous symmetry breaking and therefore $\langle M\rangle = 0$ at all T. As a result, we instead examine $\langle |M|\rangle$ and calculate the susceptibility as

$$\chi = \frac{\langle M^2\rangle - \langle |M|\rangle^2}{T}.$$

The quantities C_V/N versus T and χ/N versus T both diverge at the critical temperature T_c in the thermodynamic limit $N \to \infty$. On a finite lattice, these quantities do not diverge but will acquire peaks near T_c.

a) Use the code `plot_ising.py` to read in the Monte Carlo data from Question 1e and plot the estimators for $\langle E\rangle/N$ and $\langle |M|\rangle/N$. Explain the values you find for these estimators in the limit of small T.

b) Modify `plot_ising.py` to additionally calculate C_V and χ. Plot C_V/N and χ/N versus T and verify that there are peaks in these quantities near T_c.

c) Use `ising2d_mc.py` to generate additional data for higher L and for more temperatures close to T_c. Modify `plot_ising.py` to plot your results for several different values of L and confirm that the peaks in C_V/N and χ/N get closer to T_c as L increases.