Homework 6

Due 2019 December 6

Report instructions

Please upload *one* PDF file to Gradescope (Entry code: ME62YG).

For this homework you will implement code (on PrairieLearn) for the Ising model and simulate it in three different ways: Conventional Metropolis Monte Carlo, using the heat-bath algorithm, and a cluster algorithm.

Introduction

The 2D Ising model studied here will not be in a magnetic field and you will use periodic boundary conditions. In the absence of an external magnetic field, the only interaction in the system is the nearest-neighbor spin interaction. The Hamiltonian is:

$$H = -J \sum_{\langle i,j \rangle} s_i \cdot s_j,$$

where the summation is taken over all nearest-neighbor spin pairs in the system.

To observe the evolution of the system, you will measure the square of the magnetization per spin:

$$M^2 = \left\{ \frac{1}{N} \sum_{i=1}^{N} s_i \right\}^2 .$$

This quantity measures the amount of magnetic order in the system in a size-independent manner. If all spins are pointing up, then M^2=1. If every spin points up or down with equal probability, then M^2=0. Ideally you would study M^2 as a function of the spin-coupling strength βJ with βJ ranging from 0.0 to 1.0. If your code is fast enough, this is what you should do! Unfortunately, this may not be the case, hence, you are only required to use the following 3 values for each algorithm:

1. $\beta J = 0.3$
2. $\beta J = 0.44069$
3. $\beta J = 0.8$

Let's define a sweep as 1 step of the cluster algorithm or N steps of the Metropolis or heat bath algorithm, where N is the total number of spins in the system. For each of these algorithms, you should compute and specify the value after 1000 sweeps of

1. M^2, the square of the magnetization per spin,
2. the auto-correlation time of M^2.

To help you validate and debug your code, you should check against the result for βJ=0.25, which should lead to M^2=0.0107(1). Remember that this is the square of the magnetization *per spin*. Also recall the "comparison of data sets" section of HW1 when comparing statistical numbers.

Errors

In order to calculate the error, we typically take the standard deviation and divide by the number of effective points. For everything but the conventional metropolis at $\beta J = 0.44096$ you should use this method. For this one point the correlation time might be deceivingly large. Hence, use a different method to calculate the error of this one particular simulation: For each point (for each algorithm), run the simulation 10 times. Then calculate the standard deviation of these ten values. This should give you a more accurate assessment of your error and automatically deal with the auto-correlation time.

Initialization

For your simulations, you need to initialize your system. Do this with

```
import numpy
spinsPerSide=20
spins=numpy.ones((spinsPerSide,spinsPerSide))     # fills new array with 1
J=1.0
```

Nearest neighbors

Currently, the spins on the lattice do not know who their nearest neighbors are. Introduce them to their neighbors by constructing a neighbor list, i.e., for each spin, a list of linear indices of its neighbors. You need to do this only during the construction of the lattice and will not invoke this again during MC. Hence, speed does not matter, only correctness does.

Conventional Metropolis

The first method of simulating the Ising model will be using conventional naive Metropolis moves. Remember, the algorithm that we are proposing to do this includes:

1. Choose a random spin.
2. Flip the spin.
3. Accept with p_{accept}.

Heat-bath algorithm

Refine your code, so that it can do the heat-bath algorithm.:

1. Choose a random spin.
2. Sample a new spin for this lattice location using the heat-bath technique.
3. Accept with p_{accept}.

Note: This algorithm doesn't care what the current spin is.

If you're interested in getting an even better speedup, one way would be this (optional for everyone): Instead of flipping one spin at once, you could flip a whole block of spins at once. Imagine you take

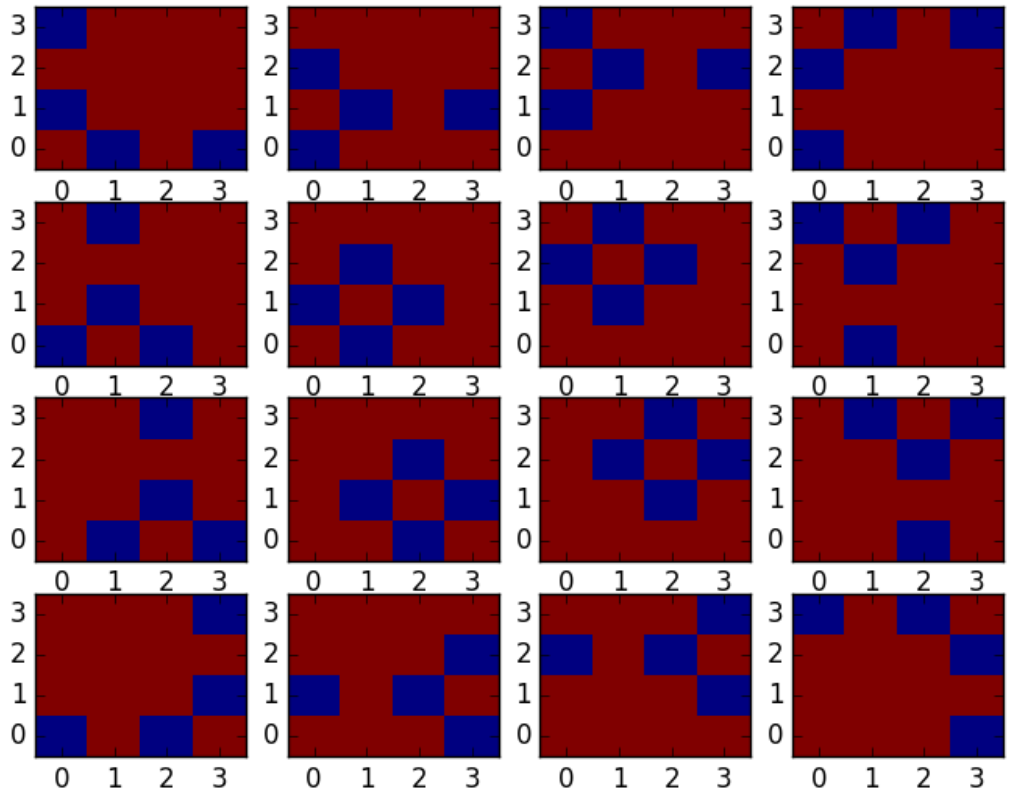

Figure 1: Correct neighbors with periodic boundary conditions should look like this.

a 3×3 block of spins and use the heat-bath technique to write down all the probabilities for all the different 2^9 spin configurations and select one with probability proportional to their likelihood.

Cluster moves

You will write code that performs a cluster move that flips a large collection of spins at once. The steps to do this will be the following:

1. Start your list of "spins you are checking" with one randomly selected spin.
2. Choose a spin off the list of spins you are checking.
3. For each neighbor which has the same spin and *which does not already belong to the cluster*
    1. add it to the cluster with probability $p = 1 - \exp(-K)$. You will solve for K later in this problem set.
    2. add it to the list of things whose neighbors you will check. (*Your code may be very slow. Find a fast way to check whether a spin doesn't belong to a cluster already.*)
4. Repeat until there are no new sites to check.

Your overall Monte-Carlo loop needs to include:

1. build a cluster,
2. flip the spins in the cluster,
3. update the current magnetization,
4. keep track of the total magnetization squared so you can average it later.

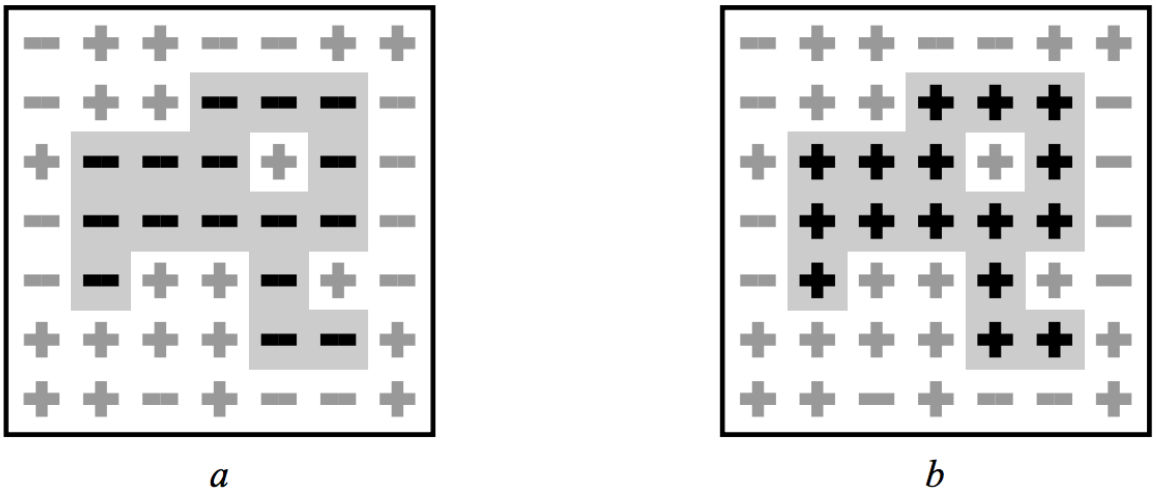

Figure 2: Illustration of a cluster flip from https://arxiv.org/pdf/cond-mat/0311623.pdf.

Your report.

Make sure you indicate clearly where you answer each question in your report. Specifically, please enter this information into Gradescope when uploading your report!

1. Write down the probability you are going to make your "new" move, i.e. $T(\text{old} \to \text{new})$, in Metropolis Monte-Carlo. Also, write down the probability you would make the "old" move given that you started in the "new" configuration, i.e. $T(\text{new} \to \text{old})$. Write down the expression for $\exp(-\delta V)$ where $\delta V = V_{\text{new}} - V_{\text{old}}$. Finally write down p_{accept}.
2. Run Monte Carlo loops, as described in the introduction above. Remember, you need to keep track of M^2 and its variance. A bad way of doing this would be to compute M^2 from scratch each time. A good way of doing this would be to update M^2 each time you flip a spin. Compute M^2 and its autocorrelation for the three temperatures discussed above!
3. Write down the probability you are going to make your "new" move, i.e. $T(\text{old} \to \text{new})$, for the heat-bath algorithm. Also, write down the probability you would make the "old" move given that you started in the "new" configuration, i.e. $T(\text{new} \to \text{old})$. Write down the expression for $\exp(-\delta V)$ where $\delta V = V_{\text{new}} - V_{\text{old}}$. Write down p_{accept}.
4. Run your heat-bath simulation and report the same information as you did for Metropolis. Is the autocorrelation time larger or smaller? Why is this?
5. Write down the probability you are going to make your "new" move, i.e. $T(\text{old} \to \text{new})$, for the cluster move. Also, write down the probability you would make the "old" move given that you started in the "new" configuration, i.e. $T(\text{new} \to \text{old})$. Write down the expression for $\exp(-\delta V)$ where $\delta V = V_{\text{new}} - V_{\text{old}}$. Write down p_{accept}. You noticed that the cluster algorithm and, thus, p_{accept} depend on a parameter, k. Write down any step of the derivation that you used, to work through the corresponding problems on PrairieLearn in the section "Cluster algorithm, determining k". Then choose k so that this turns out to be 1.0.
6. Run your cluster-move simulation and report the same information as you did before.
7. Compare your results from the different algorithms. Do they all give the same answer? If not, should you worry? Which has the least autocorrelation time for 1000 sweeps?