Physics G6080 – Problem Set 4
Fall 2022
Due Tuesday November 22, 2022

1 Monte Carlo Simulation of Liquid Argon

The Monte Carlo method can be used to simulate argon liquid, at a given temperature, through the canonical ensemble. The results of a canonical simulation should agree very closely with the results of a microcanonical simulation (molecular dynamics), differing only due to the fact that the number of particles, while large, is not infinite.

Your molecular dynamics code can be easily modified to do a Monte Carlo simulation by replacing the Verlet algorithm part of the code with a Metropolis update for each particle and velocity. Of course, you do not need to do the velocity via Monte Carlo, since it is a Maxwell distribution, but it is a negligible computational overhead to include it and it keeps the changes in your code small.

Modify your argon molecular dynamics code to do a Monte Carlo simulation at a temperature of $T = 1.069$ and density $\rho = 0.75$. Measure the same variables as in problem set 2 and check that your answers agree. Include statistical errors for your results.

In the molecular dynamics simulations, the autocorrelation times for observables are related to physical quantities, since the evolution represents real dynamics of the system. For the Monte Carlo, the autocorrelation times reflect the algorithm used for the update. Quote measured integrated autocorrelation times for the measured temperature.

2 Correlations in the 2d Ising Model

We discussed the the cluster algorithm for the 2d Ising model in class. We saw that the partition function could be written as

$$Z = \sum_{\sigma_i} \left\{ \sum_{n_{ij}} \prod_{<ij>} \left[(1-p)\delta_{n_{ij},0} + p\delta_{\sigma_i,\sigma_j}\delta_{n_{ij},1} \right] \right\} \tag{1}$$

In class, we discussed a single cluster, recursive program to simulate the Ising model and a Jupyter notebook copy of this program is posted on Canvas in
`Files -> Python Notebooks from Lectures -> Lecture 15 Ising Codes`
as `Ising Cluster.ipynb`. In this problem, you should start from this code and add measurements to measure the correlation length near the critical value of J, or $p = 1 - \exp(-2J)$.

As the Ising model approaches its critical point, the size of spatial clusters grows. It is this growth in the average size of clusters which is responsible for diverging

correlation length at T_c and the corresponding second order phase transition. The cluster algorithm avoids the critical slowing down that comes from the slow evolution through phase space of the simple, local site Metropolis algorithm.

In this problem, you should measure the spatial correlation between spins. The simplest correlator is

$$\langle \sigma(x_1, y_1)\, \sigma(x_2, y_2) \rangle \tag{2}$$

but the statistical errors are much smaller if on each configuration of an $N \times N$ lattice you calculate

$$\Sigma_x(x) = \frac{1}{N} \sum_y \sigma(x, y) \tag{3}$$

and

$$\Sigma_y(y) = \frac{1}{N} \sum_x \sigma(x, y) \tag{4}$$

and then calculate

$$\Sigma(z) = \frac{1}{2N} \left(\sum_x \Sigma_x(x) \Sigma_x(x+z) + \sum_y \Sigma_y(y) \Sigma_y(y+z) \right) \tag{5}$$

On a periodic lattice, the ensemble average $\langle \Sigma(z) \rangle$ is only a function of $|z|$. Above T_c, where there is no spontaneous magnetization, $\langle \Sigma(z) \rangle$ has the form

$$\langle \Sigma(z) \rangle = a * (\exp(-z/b) + \exp(-(N-z)/b) \tag{6}$$

2.1 Measuring this observable

Start by adding code to `Ising Cluster.ipynb` to measure the correlation in Eq. (5). Even though the cluster algorithm produces decorrelated configurations quite rapidly, you should still do some number of spin-flip updates between each measurement. 10 updates is a reasonable choice.

When you measure Eq. (5) on a configuration, you will end up with N values, one for each value of z. For each value of z, you can also determine the error on the average value, using the statistical methods of Problem Set 3.

Measure Eq. (5) on systems with values for J of 0.435, 0.430, 0.425. (Note that the critical value is $J_c = 0.4406868$ and higher temperatures correspond to lower values of J, since we wrote $J = \bar{J}/k_B T$).

2.2 Determining the correlation length

In Eq. (6) b is the correlation length, which should grow as $J \to J_c$. You can fit your data to the form of Eq. (6) to find b using the `curve_fit` function from

`scipy.optimize`. You will need to implement Eq. (6) as a function in Python and `curve_fit` will return the best fit value for b.

You should include the statistical error on each correlator point in your inputs to `curve_fit`. You don't needed to calculate the error on the returned value of the correlation length, b, but this can be done with jackknife methods.

Report on your determination of b for each of the J values you simulated. Do you find the expected change in b as $J \to J_c$?