PSet7 Report

Ali Abolhassanzadeh Mahani

December 13, 2020

1 Implementing Code to Simulate the Ising Model

I started this code by making the class `Ising` with an `__init__` function that takes in the size of the platform, L, and β, and then creates a grid of size L and fills it randomly with ± 1 as spin up/down. ($|\uparrow\rangle, |\downarrow\rangle$)

We know that the distribution for this system is canonical, of the form $Z = \sum_i e^{-\beta E_i}$. So now, I create a method `metropolis()`, which uses the metropolis algorithm to make the distribution of energy E_i to be of the canonical form. This method, loops for L^2 times and picks a cell on the grid randomly and performs the metropolis conditions on it and if it's works, the method give the cell a bit flip.

Here, in file `ising_example.py`, I initialized an Ising model of size $L = 200$ and called the `metropolis()` function on it 200 times. the results for $\beta = 0.3$ and $\beta = 0.6$ are available in Fig1

2 Data Acquisition

In order to collect data from the system, first we need a method to check if the system has reached equilibrium. In order to do that, I made the method `equalize()` that uses `metropolis()` $n = 100$ times and takes the energy samples along the process. Then checks if the auto-correlation of energies for a gap of $\frac{n}{10}$ is less that e^{-3}. If `True`, the system has reached equilibrium, otherwise repeat the process 100 more times and check for $n = 200$. and so on until we reach equilibrium.

After the system has reached equilibrium, we need to take the auto-correlation of energies to see for how many steps our data are *almost* not correlated to pick a sample. This is made possible using the `corr_len()` function. Then, I pick the data in intervals of $h =$ `corr_len(energies)`.

This process is repeated for lengths $\{100, 110, 130, 160, 200\}$ and a linear space of β of size 40, from 0.1 to 0.7 . I saved the data to CSV files and analyzed and plotted them using the file `data_analysis.ipynb`.
The plots are available in Figs 2

As for how I selected and changed the temperature, I started from high temp., $\beta = \frac{1}{k_B T} = 0.1$ to 0.7, and each time I was done taking data for each β, I the equalized state of the system with the previous value of β for the initial condition of the system with new β. This way the necessary time to reach equilibrium is decreased and so is the risk of the system freezing.

3 Finding the critical exponents ν, μ, β and the Coefficient c_0

The value for c_0 was found by fitting a log-log polynomial of degree 1 to the data. The function for c_0 is:

$$c_v = c_0 \ln(|T - T_c|), \qquad c_0 = 7.24 \pm 0.07 \tag{1}$$

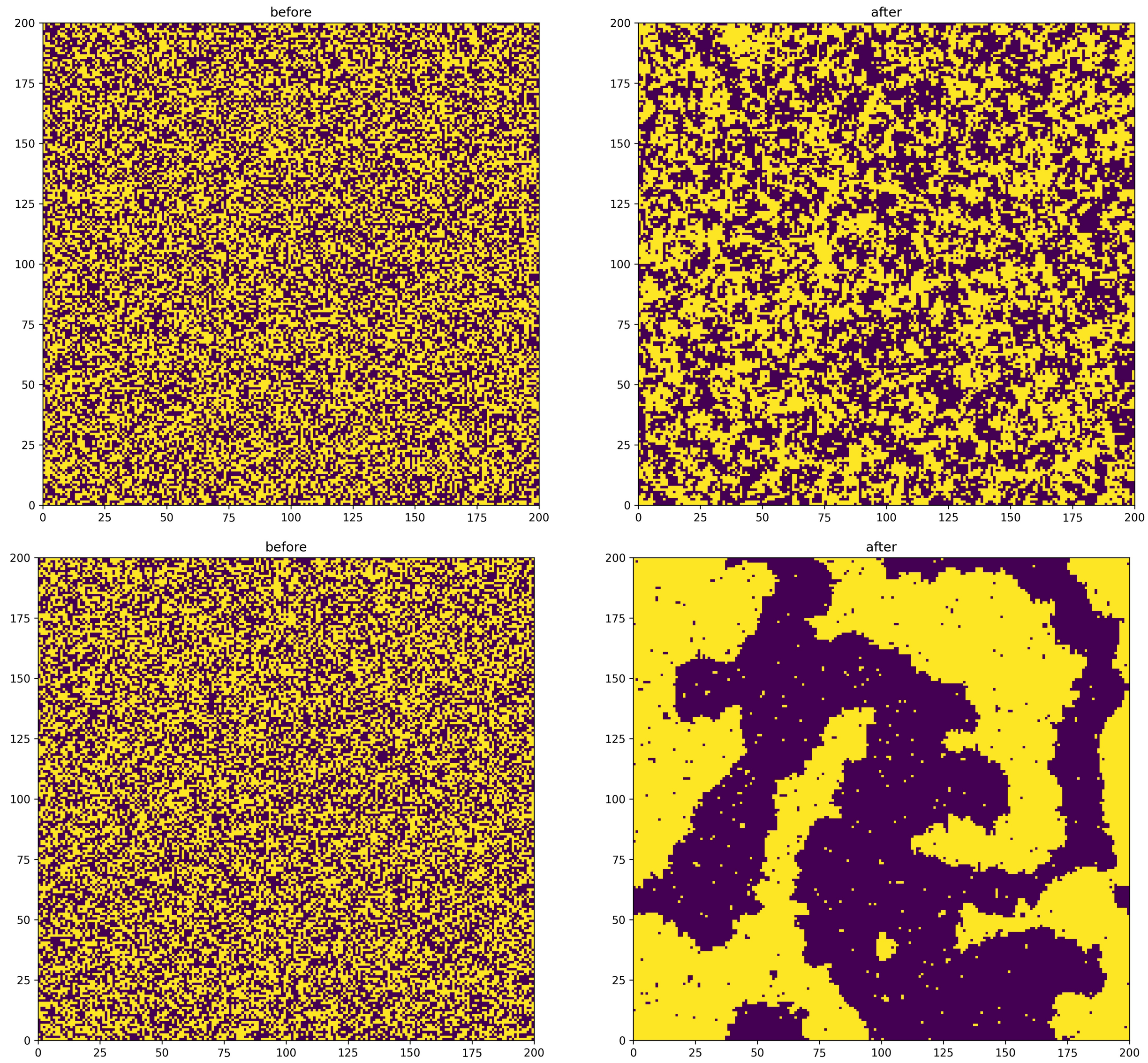

Figure 1: Ising model of size 200, before and after using the `metropolis()` function 200 times (top: $\beta = 0.3$, bottom: $\beta = 0.6$).

For the other parameters since we have already shown the plots, we'll just report the numerical values here.

$$
\begin{aligned}
m &\sim L^{-\frac{\beta}{\nu}}, & \beta &= \pm \\
\chi &\sim L^{-\frac{\gamma}{\nu}}, & \frac{\gamma}{\nu} &= -1.49 \pm 0.56 \\
C_v &\sim L^{-\frac{\alpha}{\nu}}, & \frac{\alpha}{\nu} &= -0.79 \pm 0.12 \\
\xi &\sim L^{-\frac{1}{\nu}}, & \frac{1}{\nu} &= 0.02 \pm 0.08
\end{aligned} \tag{2}
$$

4 Animating the Ising model (Bonus Question)

I finally got the hang of `plt.animation.FuncAnimation()`, I suppose. I used this to make a function `animate_ising()` in the `anime.py` file. Then I used it to animate the ising model from a uniform random distribution to equilibrium. I rendered the animations as GIFs using *imagemagick* and saved them using the value for β as a signature for the file names. (`'ising<beta>.GIF'`). The time steps are each equal to one iteration of the `Ising.metropolis()` function.

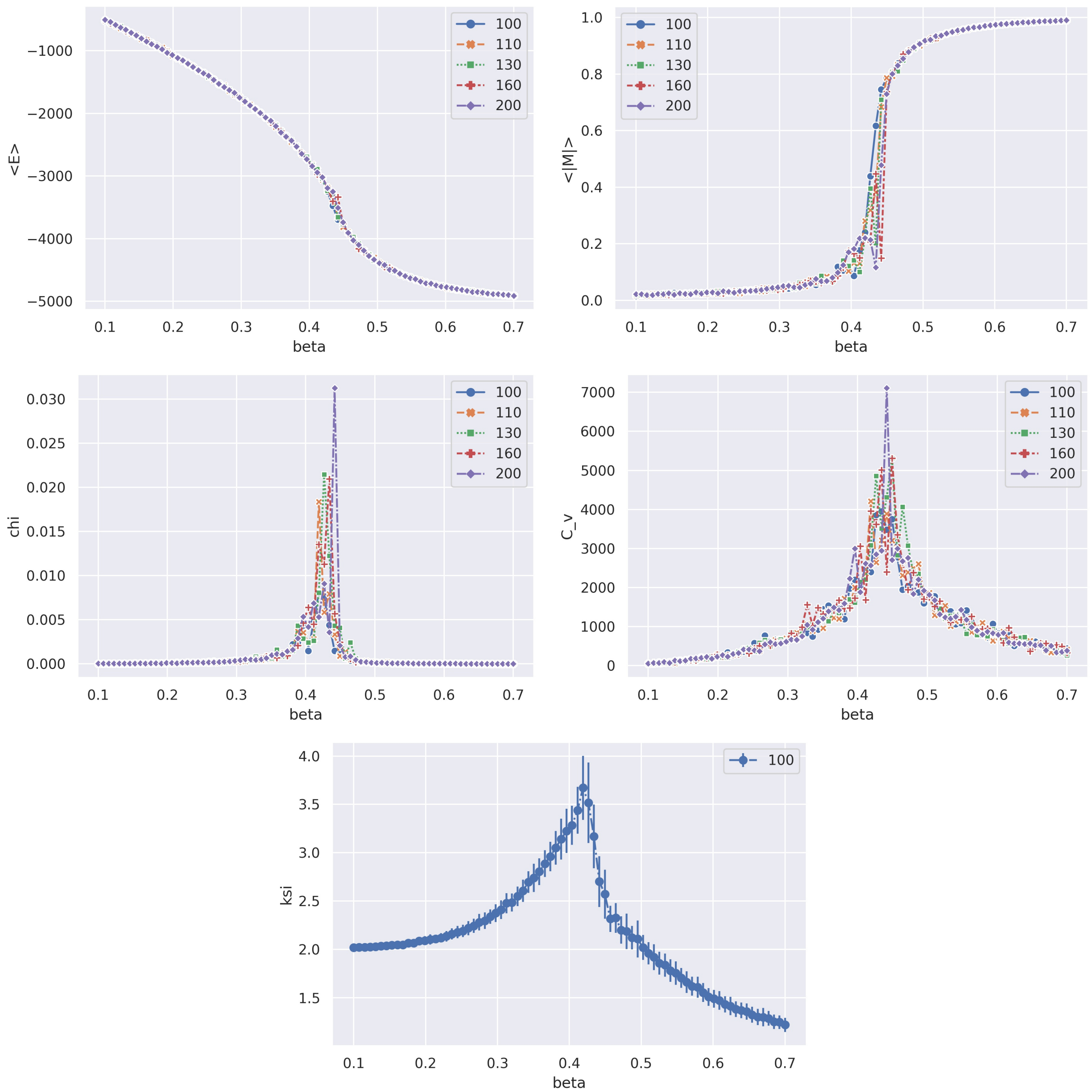

Figure 2: Data vs. beta for β from 0.1 to 0.7 and ising size in $\{100, 110, 130, 160, 200\}$. from left to right and top down, we have *energy*, *magnetization*, χ, *heat capacity* C_v, ξ *only for* $L = 100$.