Ising Model
Physics 3020 Project 1

Hailey Aronson and Finn Berquist
(Dated: October 20th, 2021)

I. OVERVIEW

In this project we investigated the magnetic properties of a two-dimensional ferromagnetic material, using a Python simulation based on the Ising model. The Ising model allows us to considers a lattice collection of many spins, which can each take the value of $+1$ or -1. This $+1$ and -1 values represent the direction of the magnetic dipole moments of the atoms within a material. Since each magnetic dipole creates is own magnetic field, each dipole will affect any other dipoles that are near it. One of the benefits of the Ising model is that as an approximation we can just consider the spin states adjacent to one another in a lattice set up. Another approximation we made in our simulation is the number of spins considered. Due to the nature of matter, this model can get very complicated very quickly. For example, one mole of atoms would be roughly 10^{26} spins to consider. For this project we examined a system of only 64 spins. Although this is a very rough approximation, it can still give us some insight into to the properties of a ferromagnetic material.

In order to find the energy and magnetization of our spin lattice, we used the following equations. The energy per spin, without double counting any neighbour interactions, is

$$E = -\frac{J}{N}\sum_{i,j}(s_{i,j}(s_{i+1,j} + s_{i,j+1})) \tag{1}$$

and the magnetization per spin is

$$M = \sum_{i,j} s_{i,j} \tag{2}$$

where $s_{i,j}$ is the spins ($+1$ or -1) at position (i, j) in an m x m matrix, $N = m^2$ is the total number of spins, and J is a constant with units of energy.

II. NUMERICAL MODEL IN 2D

For this simulation, to find average energy and magnetization for a lattice of spins of a certain size m, we used the Metropolis algorithm, which is a type of Monte Carlo algorithm. This algorithm allows us to compute states with probability $e^{\beta E_s}$, where $\beta = \frac{1}{k_b T}$, k_b is the Boltzmann constant, and T is temperature in Kelvin. The basic structure of this algorithm is:

1. Pick a random spin site and consider "flipping it", i.e., changing its current sign
2. Calculate what the new energy of the system would be, and record this change in energy as dE
3. Choose a random number between 0 and 1
4. If the random number is less than $e^{-\beta dE}$, flip
5. For a m x m matrix, repeat this process $N = m^2$ times

In our code, we created a class the "`world`" to define all of the routines we would need to run this simulation. We called our routine that implements the above algorithm, except for step 5, "`randomSweep()`". For step 5, we defined a separate routine "`nSweeps()`" so that we could call `randomSweep()` in single iterations in order to test it.

We also defined separate routines for calculating the energy and magnetization of our lattice after running the metropolis algorithm, "`calcEnergy()`" and "`calcMagnetization()`" respectively.

Since it took a little extra calculation, we also decided to define a separate routine, "`changeInEnergy()`" to find dE. By definition $dE = E_\mathrm{f} - E_0$. Since flipping a spin only changes its sign, $E_\mathrm{f} = -E_0$. Thus, $dE = -2E_0$, where E_0 is the sum of all the terms of E that include the spin site we are considering flipping. Thus, if we choose a random spin site $s_{i,j}$, since the spin of one dipole affects it's neighbours, our equation for dE will include the spin site we are considering flipping and its neighbours, so that every term with $s_{i,j}$ in it is included. For energy, without double counting, all the terms including $s_{i,j}$ are:

$$\begin{aligned} E_0 &= s_{i,j}(s_{i+1,j} + s_{i,j+1}) + s_{i-1,j}(s_{i,j} + s_{i-1,j-1}) \\ &\quad + s_{i,j-1}(s_{i,j} + s_{i-1,j+1}) \\ &= s_{i,j}s_{i+1,j} + s_{i,j}s_{i,j+1} + s_{i,j}s_{i-1,j} \\ &\quad + s_{i-1,j}s_{i-1,j-1} + s_{i,j}s_{i,j-1} + s_{i,j-1}s_{i-1,j+1} \end{aligned}$$

Now eliminating any terms not multiplied by $s_{i,j}$, we have:

$$\begin{aligned} E_0 &= s_{i,j}s_{i+1,j} + s_{i,j}s_{i,j+1} + s_{i,j}s_{i-1,j} + s_{i,j}s_{i,j-1} \\ &= s_{i,j}(s_{i+1,j} + s_{i,j+1} + s_{i-1,j} + s_{i,j-1}) \end{aligned}$$

which is just $s_{i,j}$ multiplied by each of it's neighbours!

One issue we encountered while building our algorithm, was how to account for the boundaries of the lattice while calculating energy and dE. Consider the spin site $s_{i_{max},j_{max}}$, which for simplicity we will just refer to as (i_{max}, j_{max}). The entries for $(i_{max}-1, j_{max})$ and $(i_{max}, j_{max}-1)$ are easily found, but $(i_{max}, j_{max}+1)$ and

$(i_{max}+1, j_{max})$ are not in the system because their indices are out of the grid. To account for this, we used the Python modulus operator with m, `%m`, on the coordinate we knew would be outside of our defined array, giving us the desired relevant coordinates of $(i_{max}, 0)$, and $(0, j_{max})$.

When evaluating E or dE at coordinates with $i = 0$ or $j = 0$, we do not need to use the modulus. This is because Python list indexing automatically returns the last element of a list at the index -1. So, for example, when dE is being evaluated at $(0, 0)$, the relevant neighbors are $(0, 1)$,$(1, 0)$,$(0, -1)$, and $(-1, 0)$. In python, $(0, -1) = (0, j_{max})$ and $(-1, 0) = (i_{max}, 0)$

We tested this approach with a simple 4 x 4 matrix before implementing it in our main routine to make sure it works in all cases. We also tested all of our other routines with a 3 x 3 matrix before finding the average energy and magnetization of a 64 x 64 matrix. To test the metropolis algorithm, we printed our initial matrix, ran `randomSweep()` specifying which spin to consider (instead of choosing one randomly), printed dE as well as the random number, and then checked to see if it had flipped or not, depending on what we expected it to do. To check E, M, and dE we calculated a random example by hand and compared the values with the value our routines gave us. Since we got the same values, we went ahead with $m = 64$.

III. RESULTS

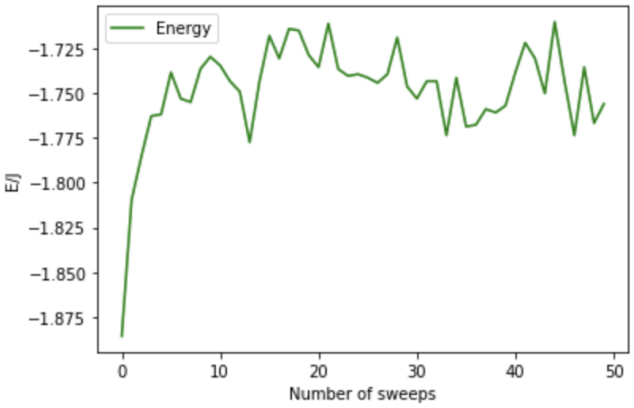

FIG. 1. One iteration of Energy per spin versus number of sweeps for m = 64 and T = 2

In figures 1 and 2 we have two plots for energy per spin as sweeps were performed on the spin system. Due to the random nature of the Metropolis Algorithm, the graph looks slightly different each time the code is run. As predicted, the values settle into an approximate equilibrium after around 20 sweeps. Counting data between the 20th and 50th sweeps then, we found an average energy/J per spin to be -1.7548314144736843, with a standard deviation of 0.024103646829361593.

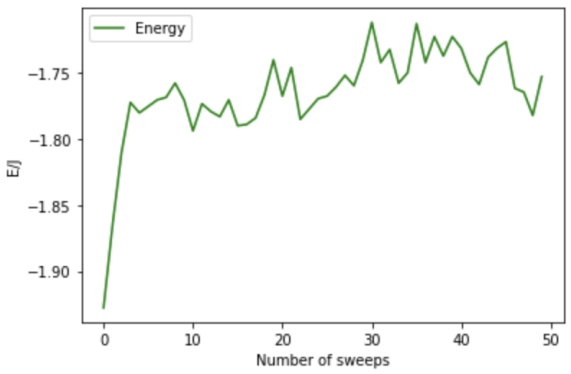

FIG. 2. Another iteration of Energy per spin versus number of sweeps for m = 64 and T = 2

In figures 3 and 4 we have two plots for magnetization per spin as a function of sweeps. Again recording data just for sweeps 20-50, we found an average magnetization per spin of 0.9154502467105263, with a standard deviation of 0.00770130436080275.

Next, we gathered data about average magnetization and energy per spin at different temperatures. We calculated these averages by running a spin simulation that calculated the energies and magnetization's after each spin in the relevant equilibrium region(spins 20-50), as is represented in figures 5 and 6. Note that the averages found above appear on these graphs for T = 2.

We can see in figure 5 that as temperature increases, the average energy per spin also increases. This is what we expect, since temperature is a form of energy, so as the temperature rises, the energy of the system should also increase. In figure 6, we can see that at 3 Kelvin, the system encounters its critical temperature, or the temperature at which our collection of spins is no longer likely to magnetize. Although magnetization is more likely at low temperatures in general, this is quite a low critical

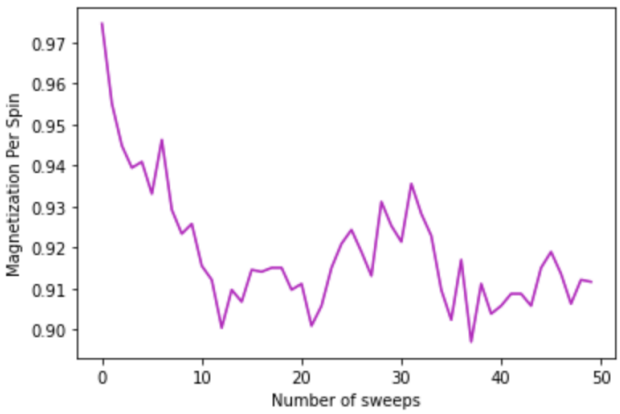

FIG. 3. One iteration of Magnetization per spin versus number of sweeps for m = 64 and T = 2

temperature, especially considering room temperature is about 289 degrees Kelvin, and we know that there exist magnets that are magnetized at room temperature. This may be because we are only considering a collection of 64 spins, which is a very approximation.

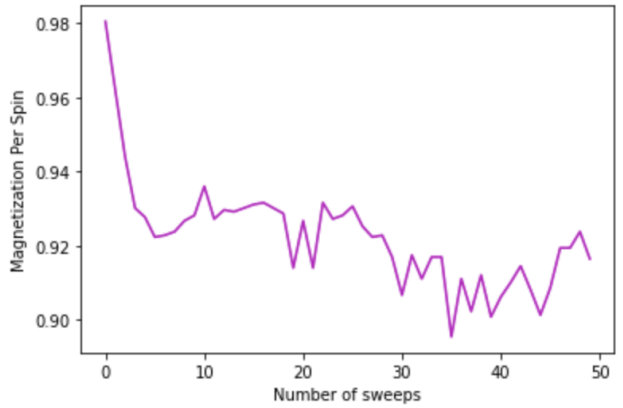

FIG. 4. Another iteration of Magnetization per spin versus number of sweeps for m = 64 and T = 2

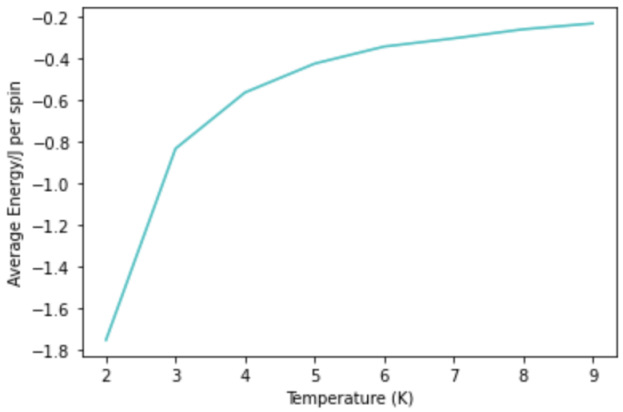

FIG. 5. Energy versus temperature for sweeps 20-50

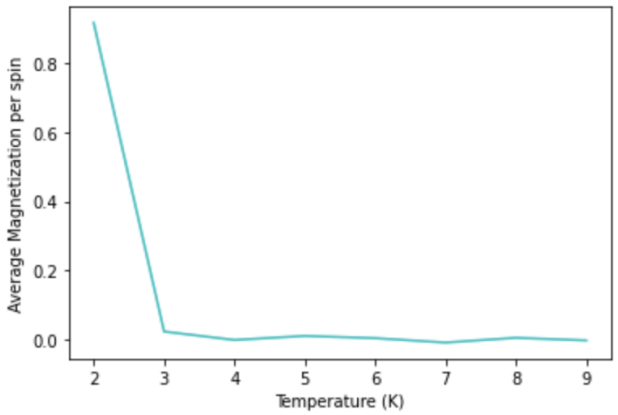

FIG. 6. Magnetization versus temperature for sweeps 20-50