Monte-Carlo simulations of a 2D Ising model

Joe Hart - CID 01852112

1 Introduction

The Ising model is a physical model originally developed to model ferromagnets, but now, due to its simplicity and adaptability, has been applied in many other fields, such as modelling binary alloys and protein folding.[1] The Ising model involves a lattice of 1, 2 or 3-dimensions where each site has a spin that is either "up" (+1) or "down" (-1). Figure 1 demonstrates Ising lattices in 3 different dimensions. The 2D Ising model is one of the simplest models to show a phase transition, and it can be used to predict physical quantities such as specific heat, correlation lengths and many more.[2]
Analytically solving the Ising model is only possible for simple cases such as the one-dimensional case, solved by Ising in 1925,[3] and the two-dimensional case in the absence of an external field,[4] with no analytical solutions known for dimensions higher than two. We will be solving the two-dimensional case in the absence of a field. Although this is analytically solvable, the solutions are complex, meaning the preferred route is to attack the problem computationally with Monte-Carlo methods. We will be using the Metropolis algorithm, an algorithm that changes a single spin per step to produce a new state, accepting or rejecting the new state based on a probability distribution, which will be covered in more depth in Task 3b.

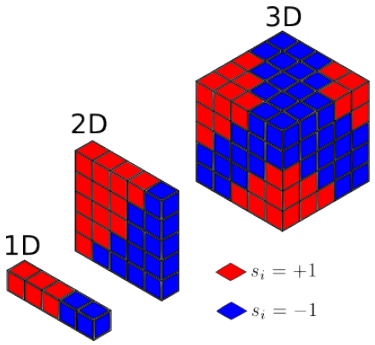


Figure 1: Lattices of 1, 2 and 3-dimensions with colour-coded spins[5]

This investigation will study ferromagnetism and the transition from ferromagnetism to paramagnetism. Ferromagnets are materials which exhibit spontaneous magnetisation due to the alignment of their magnetic dipole moments, resulting in a net magnetic moment in the absence of a field. A material that has no preferential magnetic dipole alignment, so that their dipoles have random orientations, has a zero net magnetic moment and are known as paramagnetic.
Ferromagnetism is the result of the energy minimisation winning competition between energy minimisation and entropy maximisation. Parallel alignment of spins (ferromagnetism) is energetically favourable but increases the order of the system, leading to a decrease in entropy. The favoured state determines if the material will exhibit ferromagnetism (energy minimisation dominates: spins parallel therefore net magnetic moment $\neq 0$) or paramagnetism (entropy maximisation dominates: spins disordered therefore zero net magnetic moment). The winner of this competition depends on the temperature of the system, as there is an energy barrier associated with flipping a spin due to the unfavourable nearest neighbour interactions caused by anti-parallel spins.
The Curie temperature, T_C, is the critical temperature at which a material undergoes a second order phase transition from an ordered ferromagnetic state to a disordered paramagnetic state.[6] Above the Curie temperature, the thermal energy is sufficient to overcome the energy barrier associated with flipping a spin, and the system is now dominated by entropy maximisation. The configuration that maximises the entropy is when a spin to be anti-parallel with all its nearest neighbours, resulting in a net zero magnetic moment.

2 Task 1

In the absence of a magnetic field, the nearest neighbour spin-spin interactions are the only contributions to the energy. The total interaction energy is defined as:

$$E = -\frac{1}{2}J\sum_{i}^{N}\sum_{j\ \in\ \text{neighbours}(i)} s_i s_j \tag{1}$$

Where the constant J dictates the strength of interaction, and the factor of a half prevents double counting of spin-spin interactions (as spin i interacting with spin j is equivalent to spin j interacting with spin i). We are working in reduced units where J = k_B = 1. As periodic boundary conditions are used, if a spin is in the "end" position of the lattice, the corresponding spins at the "start" in each direction will count as its nearest neighbours too.

2.1 Task 1a

Starting with the case of energy minimisation. Energy minimisation occurs when all the spins are aligned, therefore $s_i = s_j = \pm 1$, therefore there are two microstates where the energy is minimised. As in this case $s_i s_j = 1$, We are able to simplify Equation 1:

$$E = -\frac{1}{2}J\sum_{i}^{N}\sum_{j\ \in\ \text{neighbours}(i)} 1 \tag{2}$$

The sum of nearest neighbours will depend on the dimension D. Looking at Figure 1 we can see that each spin will have 2D nearest neighbours:

$$E = -\frac{1}{2}J\sum_{i}^{N} 2D \tag{3}$$

The sum of 2D from i to N is just equal to 2DN, which gives us the expression for the lowest possible energy configuraiton for the Ising model:

$$E = -DNJ \tag{4}$$

As there are 2 microstates for the lowest energy configuration, Ω=2 and the entropy therefore equals (k_B = 1):

$$S = k_B ln(\Omega) = k_B ln(2) = ln(2) \tag{5}$$

2.2 Task 1b

We are now able to calculate the lowest energy configuration for the Ising model just by knowing the dimension, number of particles and J. Example system: D = 3, N = 1000, J = 1

$$E = -(3)(1000)(1) = -3000K \tag{6}$$

Flipping a spin will increase the energy of the system. As each spin is only effected by its nearest neighbours, flipping a single spin in our 3D system will only effect 6 spins. As we are just looking at one spins interactions with its neighbours, there is no double counting and the factor of a 1/2 does not need to be included:

$$E = -J\sum_{i=1}^{1}\sum_{j\ \in\ \text{neighbours}(i=1)} s_i s_j = 6J \tag{7}$$

Therefore instead of one spin reducing the energy by 6J, it increases the energy by 6J: $\Delta E = +12$J. The energy may have been reduced, but the disorder of the spins has increased and therefore the entropy has increased. There are 2 microstates for the lowest energy configuration, therefore $\Omega_1 = 2$. For the new configuration, the number of microstates, Ω_2, is:

$$\Omega_2 \frac{N!}{n!(N-n)!} = 2\frac{1000!}{(1!)(999!)} = 2000 \tag{8}$$

Therefore the change in entropy is:

$$\Delta S = k_B ln(\Omega_2) - k_B ln(\Omega_1) = k_B ln(\frac{\Omega_2}{\Omega_1}) = k_B ln(\frac{2000}{2}) = k_B ln(1000) = ln(1000) \tag{9}$$

Using $k_B = 1$

2.3 Task 1c

A key quantity in the system is the total magnetisation, which is calculated by summing the spins:

$$M = \sum_i s_i \tag{10}$$

For example, for the 1D and 2D lattices in Figure 1:

$$1D : M = 3 - 2 = +1 \tag{11}$$

$$2D : M = 13 - 12 = +1 \tag{12}$$

The magnetisation depends on the temperature as at low temperatures the thermal energy is too low to overcome the energy barrier to flip a spin. For example, an Ising lattice with D = 3, N = 1000 at T = 0 K would be expected to have all its spins aligned as the thermal energy is zero, and therefore a net magnetisation of ± 1000 (all spins up or all spins down).

3 Task 2

3.1 Task 2a

For larger lattice sizes, the calculations of energy and magnetisation are much more easily done in python. The following function was used for lattice energy calculations:

```
def energy(self):
    "Return the total energy of the current lattice configuration."
    energy = 0.0

    for i in range(self.n_rows):
        for j in range(self.n_cols):
            sum_sisj = 0

            #periodic boundary conditions using remainders
            R = (i+1)%self.n_rows
            L = (i-1)%self.n_rows
            UP = (j+1)%self.n_cols
            DOWN = (j-1)%self.n_cols

            # Array of neighbouring spins in the lattice
            neighbours = np.array([self.lattice[R, j], self.lattice[L, j],
            self.lattice[i, UP], self.lattice[i, DOWN]])

            # Compute the sum of neighbouring spins
            sum_sisj = np.sum(self.lattice[i, j] * neighbours)
            energy += -0.5*self.J*sum_sisj
    return energy
```

The following function was used for Magnetisation calculations:

```
def magnetisation(self):
    "Return the total magnetisation of the current lattice configuration."
    magnetisation = np.sum(self.lattice) # sum the spins in the lattice
    return magnetisation
```

3.2 Task 2b

A check was performed to test if the energy and magnetisation functions calculated the correct values for 3 test lattices shown in Figure 2.

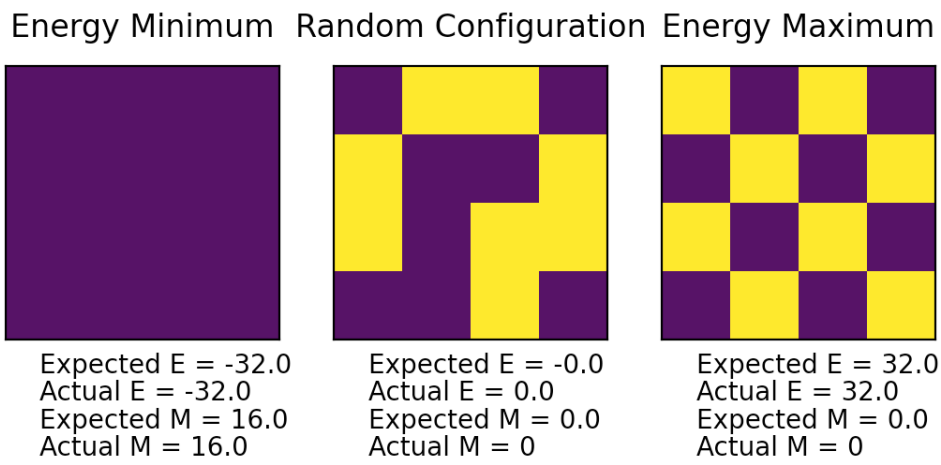


Figure 2: Calculated vs expected values for 3 test cases, showing that the functions written produced the expected values.

4 Task 3

4.1 Task 3a

Large systems can have an extremely large number of configurations, which is very computationally expensive to calculate the average energies/magnetisation for. For example, for a system with 100 spins and a computational power allowing analysis of $1x10^9$ configurations per seconds:

$$\text{Number of configurations} = 2^{100} \tag{13}$$

$$\text{Time to compute average magnetisation} = \frac{2^{100}}{10^9} = 1.3 \times 10^{21} s = 4 \times 10^{13} \text{ years} \tag{14}$$

This is therefore not a viable method. The contribution of a microstate to the system is determined by the Boltzmann factor, which for the majority of the configuration will be very small. This means by considering only configurations that have major contributions to the system averages we are able to save time and make this calculation possible. This is importance sampling, where the only points in space we sample are the points which the system is likely to occupy.

4.2 Task 3b

In order to implement importance sampling, the Metropolis Monte-Carlo method is used. This algorithm works via creating a new configuration by randomly choosing one spin and flipping it. If there energy of the new state is lower than the old one, it is automatically accepted as the new configuration. If the energy is higher, the new configuration is accepted according to the Boltzmann distribution by comparison with a random number R in the interval [0,1):

$$R \leq exp\left(-\frac{\Delta E}{k_B T}\right) \text{ Accept} \tag{15}$$

$$R > exp\left(-\frac{\Delta E}{k_B T}\right) \text{ Reject} \tag{16}$$

The algorithm was implemented into python as the following method: The following method for a single Monte-Carlo step (flipping a single spin) was implemented into python:

```
def montecarlostep(self, T):
    #update no. cycles
    self.n_cycles += 1

    # define the energy of the old state
    energy = self.energy()

    # Select the coordinates of the random spin
    random_i = np.random.choice(range(0, self.n_rows))
```

```
        random_j = np.random.choice(range(0, self.n_cols))

        # flip the spin
        self.lattice[random_i, random_j] = -1*self.lattice[random_i, random_j]

        # calc energy of new config
        new_energy = self.energy()
        E_diff = new_energy - energy

        # choose a random number in the range[0,1)
        random_number = np.random.random()

        # probability distribution being used
        botlzmann = np.exp(-E_diff/(T))

        if (E_diff > 0 and random_number > botlzmann):
            # reject
            self.lattice[random_i, random_j] = -1*self.lattice[random_i, random_j]

        else:
            # accept the copy and return the new config and magnetisation
            energy = new_energy

        M = self.magnetisation()

        #update values
        self.E += energy
        self.E2 += energy**2
        self.M += M
        self.M2 += M**2

        return energy, M
```

Breakdown of steps in method: record energy of current state → flip a single spin at random → calculate new energy → if the energy difference is greater than zero and the Botlzmann factor is greater than R, reject the configuration by changing back the spin → else keep the new configuration and change the energy to be the energy of the new configuration → update running total of quantities.
A single energy measurement is not very reliable and may not have much meaning. It is important to calculate averages of quantities over a certain number of steps to understand what is happening in the system. The statistics method, whenever called, calculates the average E, E^2, M and M^2 over all the number of steps that have occurred so far, also returning the number of steps:

```
    def statistics(self):
        N = self.n_cycles

        # find the avg by diving by the number of cycles
        avg_E = self.E/N
        avg_E2 = self.E2/N
        avg_M = self.M/N
        avg_M2 = self.M2/N

        return avg_E, avg_E2, avg_M, avg_M2, N
```

4.3 Task 3c

The python script ILamin.py was used to run just under 1500 Monte-Carlo steps for an 8x8 lattice at T = 1K to study its approach to equilibrium. Figure 3 shows the system reaching equilibrium from its initial random configuration. This simulation was done at T< T_C. If T< T_C, spontaneous magnetisation is expected as in this temperature range energy minimisation is favoured. This means that the alignment of spins is favoured over random spins and the system is expected to have a non-zero net magnetisation. The system reaches equilibrium when its E and M stops changing, but for this to not be a metastable state all the spins must also be aligned in the lattice image above the graph (spins are colour coded, so

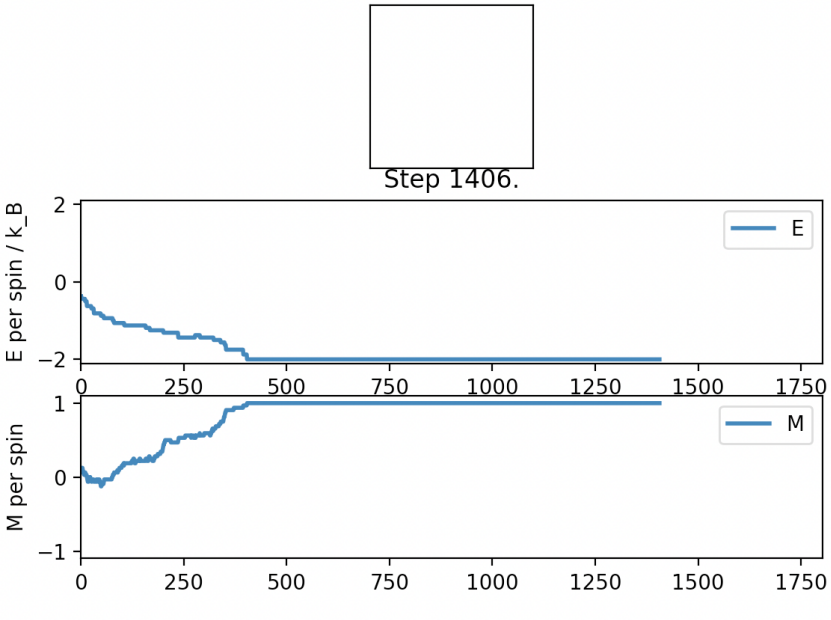


Figure 3: Monte-Carlo simulation for the Ising model with an 8x8 lattice at T = 1K

Table 1: Average quantities produced from the Monte-Carlo simulation.

E(K)	$E^2(K^2)$	M	M^2
−1.75	3.06	0.83	0.69

the spins in Figure 3 are all aligned). As expected, the magnetisation per spin = 1 (or -1 depending on the direction of the aligned spins) and the energy per spin = -2 as the spins are all aligned. Table 1 shows the average values calculated using the statistics method. These values are not what is expected for a system at equilibrium. This error is due to averaging over all the data, even before it reaches equilibrium, instead of only once the system has reached equilibrium.

5 Task 4

5.1 Task 4a

The speed of these simulations is fast for a small number of steps, however, the code will struggle for a large number of steps, looping through a temperature range. Using the current energy() and magnetisation() methods defined in task 2a, it takes 23 ± 0.09 s to run 2000 Monte-Carlo steps. A larger lattice such as 32x32 has 1024 spins, meaning 2000 steps does not even flip all the spins twice. This slow computation speed will be detrimental to recording data at a suitable timescale for larger lattices and needs to be improved.

5.2 Task 4b

In order to improve computation speed, we need to do three things: minimise the number of times methods are called, minimise for loops, and use Numpy functions where suitable. Many Numpy operations are implemented in C, greatly increasing speed.[7] The Magnetisation method used already uses Numpy's np.sum function, meaning no optimisation is needed. However, the energy method uses a double for loop to find the nearest neighours for each spin, one at a time. Numpy's roll function makes it possible to implement periodic boundary conditions without a for loop, for the entire lattice in one step. To reduce the number of calculations, it is possible to only shift the lattice to the right and upwards (or any combination of an x shift + y shift) instead of in all 4 directions to avoid double counting, meaning the factor of 1/2 is not needed in the energy calculation:

```
def energy(self):
    """
    Return the total energy of the current lattice configuration.
```

```
        """
        # np.roll to shift the entire lattice in the specified direction
        # (periodic boundary conditions)
        # Just 2 directions to avoid double counting (and to do less calculations)
        R = np.roll(self.lattice, shift=-1, axis=0)
        UP = np.roll(self.lattice, shift=-1, axis=1)

        # Compute the sum of neighbouring spins
        # np.multiply is equivalent to *
        sum_sisj = np.sum(self.lattice * (R + UP), axis=(0, 1))

        # Compute and return the energy
        energy = -self.J * sum_sisj
        return energy
```

5.3 Task 4c

Running 2000 Monte-Carlo steps with the optimised program took 0.1721 ± 0.0020 s, over 100x faster than the previous code (23 s).

6 Task 5

6.1 Task 5a

In order to correct the averages produced in Task 3c, a method of determining the number of steps a system takes to reach equilibrium must be devised. We want to find the number of steps as a function of grid size. To create this function, Monte-Carlo simulations were performed with repeats for a range of lattice sizes, and the number of steps to reach equilibrium was estimated. The number of steps was estimated by looking at the simulations energy and magnetisation profiles as well as the spins, shown in Figure 4. Once all the spins were aligned and the energy and magnetisation stable, the system was said to be at equilibrium.

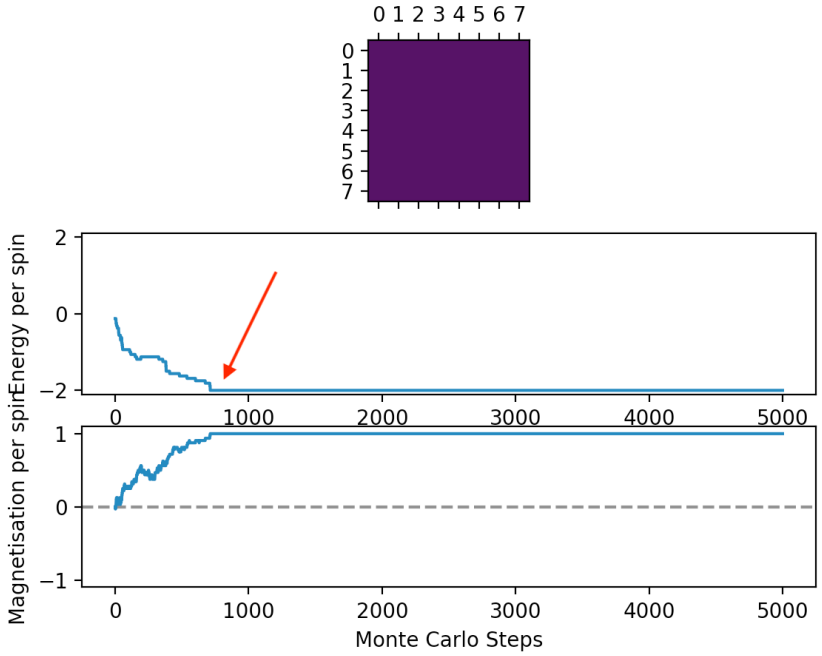


Figure 4: System at equilibrium, with the point at which equilibrium was reached highlighed by the red arrow

The estimated number of steps were plotted against grid size in Figure 5. The relationship was expected to be quadratic, however the fitting was quite poor, meaning the relationship was more complex than first thought.
After some experimentation, it was found that a log-log plot yielded a linear relationship (Figure 6a), therefore fitting the curve to y = mx + c would make determination of the relationship simple.

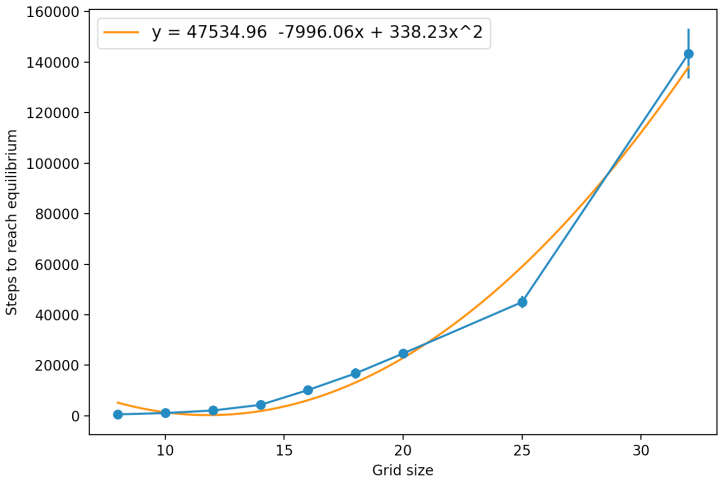
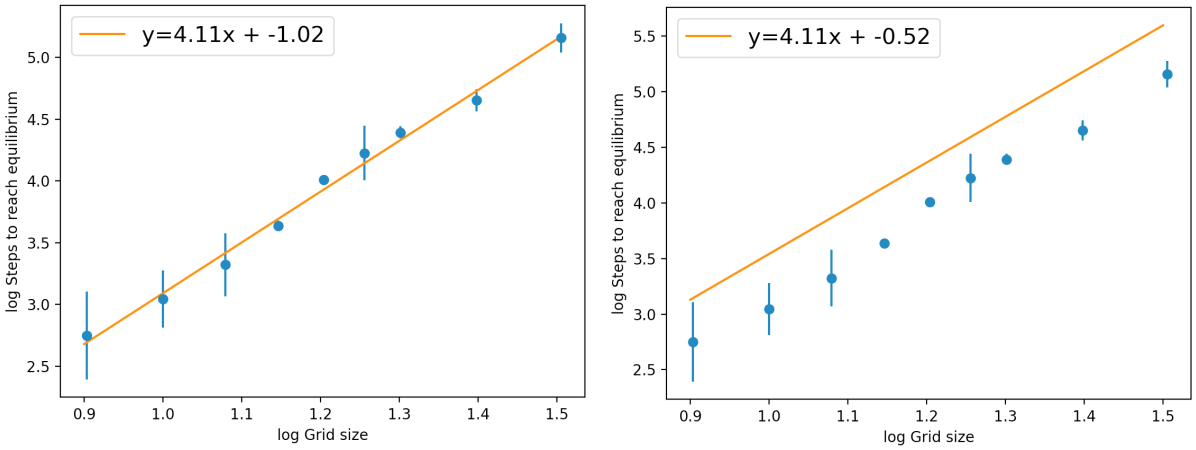


Figure 5: Fit using polyfit with polynomial degree = 2



(a) Log-log plot

(b) A Shifted fit of log-log plot

Figure 6: New log-log plot fits

So far we have the fitted the averages of the number of steps to reach equilibrium, however, in order to make sure the system is definitely at equilibrium, the fit must be shifted above the error bars of the points. If we left the fit how it is, when we record energy and magnetisation averages the system may not always be at equilibrium, which would be problematic for calculating average values. Figure 6b shows the shifted fit. In order to get a cuttoff function for the number of steps to reach equilibrium to use in our code, both sides of the shifted fit equation must be raised to the power of 10 to reveal the following relationship, where a Monte-Carlo cycle is completed when every spin in the lattice has been flipped:

$$\text{Monte-Carlo steps to reach equilibrium, N} = 10^{-0.52}\text{grid size}^{4.11} \tag{17}$$

$$\text{Monte-Carlo cycles to reach equilibrium, N} = 10^{-0.52}\text{grid size}^{2.11} \tag{18}$$

The procedure was repeated for different temperatures (Figure 7), however, the difference in cutoff value was negligible, and will be ignored for the purpose of this experiment. In order to yield accurate average values, the cuttoff was implemented into the statistics() and montecarlostep() methods to ignore steps before equilibrium has been reached:

```
cutoff defined within __init__:
    def f(x, a, b, c):
        return a + 10*np.exp(c*x)

    # 10**(-0.52 + 4.11*log10(n_rows) = 10**(-0.52) * n_rows**(4.11)
    self.cutoff = 10**(-0.52) * n_rows**(4.11)
```

```
def montecarlostep(self, T):
    .
    .
    .
    # end of code changed:
    if self.n_cycles > self.cutoff:
        self.E += energy
        self.E2 += energy**2
        self.M += M
        self.M2 += M**2
    else:
        pass
    return energy, M

def statistics(self):
    N = self.n_cycles
    cutoff = self.cutoff
    if N > cutoff:
        avg_E = self.E/(N-cutoff)
        avg_E2 = self.E2/(N-cutoff)
        avg_M = self.M/(N-cutoff)
        avg_M2 = self.M2/(N-cutoff)

    return avg_E, avg_E2, avg_M, avg_M2, N
```

Table 2 shows the new average values, which align with our expectations very well. An M of effectively 1 shows that all the spins are aligned and the material is ferromagnetic, which is expected at this temperature. The energy value also agrees with this, averaging as each spin stabilising the energy by 2 K, the expected value.

Table 2: New averaged quantities

E(K)	$E^2(K^2)$	M	M^2
−1.999	3.989	0.999	0.998

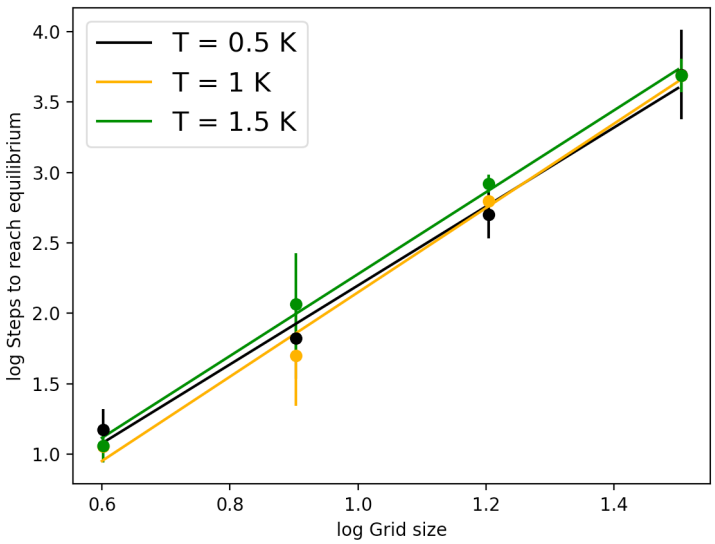


Figure 7: Testing at different temperatures. Smaller grid sizes were used to reduce computation time

6.2 Task 5b

So far we have only investigated energy and magnetisation at a single temperature. Figure 8 shows the the energy and magnetisation per spin for an 8x8 lattice from 0.5 to 5 K. The number of cycles required

to run was calculated with Equation 17, however, it was found that the longer the simulations were run, the better the quality of results. This means that for smaller lattices such as 8x8 where running many steps is not particularly computationally expensive, so around 50000 steps were run. This also gave any lattices stuck in metastable states more time to equilibrate, which was most likely the reason for higher quality results.

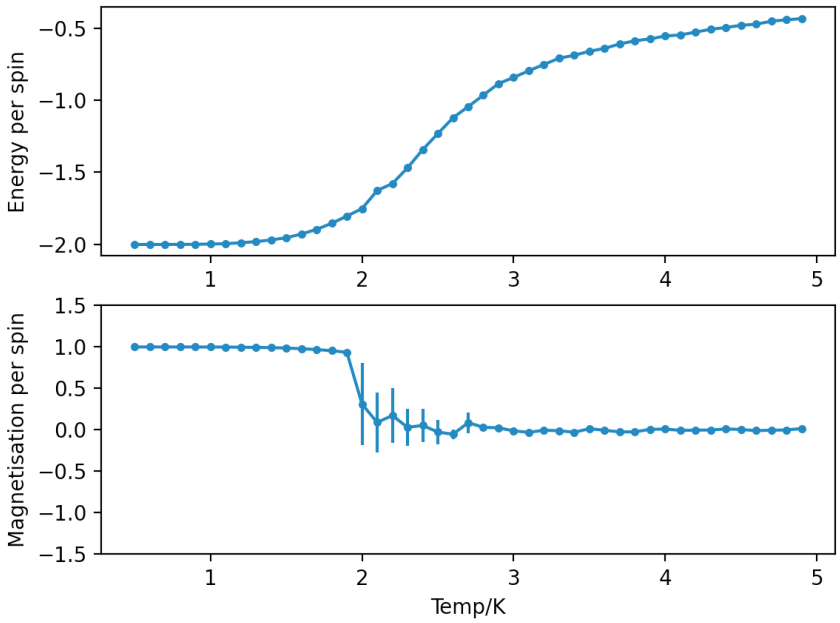


Figure 8: 8x8 lattice over a T range of 0.5-5 K with an interval of 0.1 K

The magnetisation is initially at +1, but upon increasing the temperature a transition occurs where the value drops and then oscillates about zero. This shows the phase transition from ferromagnetic to paramagnetic as the high temperature state has no net magnetic moment, meaning is has transitioned to a paramagnetic state of randomly orientated spins. The Curie temperature is in this transition range and we will find it in Task 8. The errors in the energy per spin is minimal, but in the magnetisation there is a significant increase in the error around the Curie temperature, characteristic of this phase transition. This is due to large fluctuations in the magnetisation as it is approaching the Curie temperature as its correlation length is increasing (Until it diverges at T_C). In this region around the T_C, there is a fine balance between energy minimisation and entropy maximisation which also contributes to the fluctuations of magnetisation. To observe these fluctuations, the data must be recorded with smaller temperature increments, which we will observe in task 8 with the C++ data as the computational cost is too large in python. These fluctuations are caused due to the competition between energy minimisation and entropy maximisation is so close near the critical temperature.

7 Task 6

Task 5b was repeated for lattice sizes ranging from 2x2 to 32x32, shown in Figure 9. The results produced by smaller lattices are poor as spins "interact with themselves" because of their close proximity to themselves due to the periodic boundary conditions. These smaller lattices have larger error bars as their values of magnetisation fluctuate more. This is because in a small grid the effect of changing one spin is more dramatic than in a larger grid leading to more variation. The larger the lattice we work with, the more reliable and consistent the results. Looking at the energy, it would be expected that above the critical temperature the energy should be zero as the net magnetisation is zero. You can see that as lattice size increases, the energy gets closer to zero, however, at the larger lattices there is less change upon increase in size (e.g. between 16 and 32). This suggests that some residual correlation between neighbouring spins is still present even above the curie temperature, which our model does not account for.[8]

For magnetisation, increasing the grid size results in a less gradual and more accurately represented phase transition which fluctuates less. All lattice sizes have characteristic larger magnetisation errors around the phase transition.

Comparing the differences between the energy and magnetisation curves for different lattice sizes, we can determine the size at which the accuracy gained from increasing the lattice size is no longer worth the

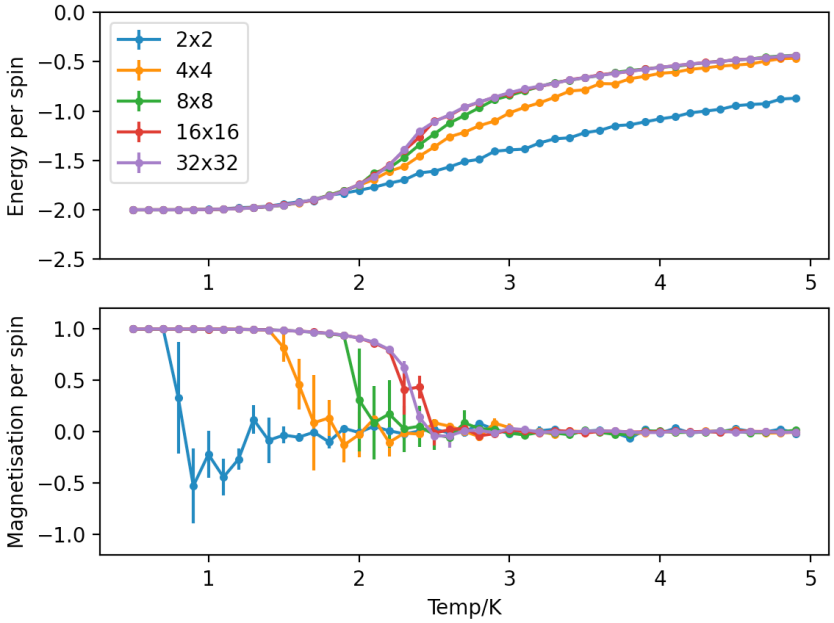


Figure 9: Energy and magnetisation per spin for a range of lattice sizes. The legend applies to both plots.

extra computation time. This is the case when looking at the curves for 16x16 and 32x32, which do not show significant differences, especially when compared to differences between the smaller grid sizes. This is also the lattice size which is large enough to capture the long range interactions and for its spins to not have a major effect on themselves, such as in a 2x2 grid where this is a major source of error.

8 Task 7

8.1 Task 7a

In task 6 we were able to observe a phase transition, but not pinpoint it. In order to do this we can study the heat capacity, which diverges at the critical temperature. The heat capacity is defined as:

$$C = \frac{\partial \langle E \rangle}{\partial T} \tag{19}$$

This is not a very convenient form, so we can rewrite it by performing the derivative and using the definition of $\langle E \rangle$:

- The mean energy is defined as follows:

$$\langle E \rangle = -\frac{1}{Z}\frac{\partial Z}{\partial \beta} \tag{20}$$

- Where Z is the partition function:

$$Z = \sum e^{-\beta E} \tag{21}$$

- Subbing $\langle E \rangle$ into the equation for C:

$$\frac{\partial \langle E \rangle}{\partial T} = \frac{\partial}{\partial T}\left(-\frac{1}{Z}\frac{\partial Z}{\partial \beta}\right) \tag{22}$$

- Rewriting the derivative:

$$\frac{\partial \langle E \rangle}{\partial T} = \frac{\partial \beta}{\partial T}\frac{\partial}{\partial \beta}\left(-\frac{1}{Z}\frac{\partial Z}{\partial \beta}\right) \tag{23}$$

- Applying $\frac{\partial}{\partial \beta}$:

$$\frac{\partial \langle E \rangle}{\partial T} = \frac{\partial \beta}{\partial T}\left(\frac{\partial Z}{\partial \beta}\frac{1}{Z^2}\frac{\partial Z}{\partial \beta} - \frac{1}{Z}\frac{\partial^2 Z}{\partial \beta^2}\right) \tag{24}$$

- As $\beta = \frac{1}{k_B T}$, $\frac{\partial \beta}{\partial T} = -\frac{1}{k_B T^2}$:

$$\frac{\partial \langle E \rangle}{\partial T} = -\frac{1}{k_B T^2} \left(\left(\frac{1}{Z} \frac{\partial Z}{\partial \beta} \right)^2 - \frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2} \right) \tag{25}$$

- LHS term is familiar:

$$\left(\frac{1}{Z} \frac{\partial Z}{\partial \beta} \right)^2 = \langle E \rangle^2 \tag{26}$$

- Concentrating on the RHS term:

$$\frac{\partial Z}{\partial \beta} = -\sum E e^{-\beta E} \text{ and } \frac{\partial^2 Z}{\partial \beta^2} = \sum E^2 e^{-\beta E} \tag{27}$$

$$\frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2} = \frac{1}{Z} \sum E^2 e^{-\beta E} \tag{28}$$

- For any observable A, the ensemble average is:

$$\langle A \rangle = \sum A \rho(\alpha), \text{ where } \rho(\alpha) \text{ is a probability distribution} \tag{29}$$

- Here our probability distribution is the Botlzmann distribution, therefore we can write Equation 28 as an average value:

$$\frac{1}{Z} \frac{\partial^2 Z}{\partial \beta^2} = \frac{1}{Z} \sum E^2 e^{-\beta E} = \langle E^2 \rangle \tag{30}$$

- We can now simplify our equation:

$$\frac{\partial \langle E \rangle}{\partial T} = -\frac{1}{k_B T^2} \left(\langle E \rangle^2 - \langle E^2 \rangle \right) \tag{31}$$

- Noticing the parallel with variance:

$$Var[E] = E[E^2] - E[E]^2 \tag{32}$$

$$Var[E] = \langle E^2 \rangle - \langle E \rangle^2 \tag{33}$$

- We have a new expression for heat capacity, C:

$$C = \frac{\partial \langle E \rangle}{\partial T} = \frac{Var[E]}{k_B T^2} \tag{34}$$

8.2 Task 7b

Plotting the heat capacity for grid sizes ranging from 2x2 to 32x32 (Figure 10), we observe that as the gridsize is increased, the peak becomes increasingly sharp at T_C. This is because as the grid size tends to infinity, the heat capacity at the T_C also tends to infinity. However, we do not observe the divergence of T_C due to the finite size effect. For a small grid, the long range correlations cannot be modelled and therefore the heat capacity is only weakly peaked at T_C. Looking at the position of the peaks, the T_C seems to be decreasing with increasing grid size, tending to the value of T_C for an infinite lattice.

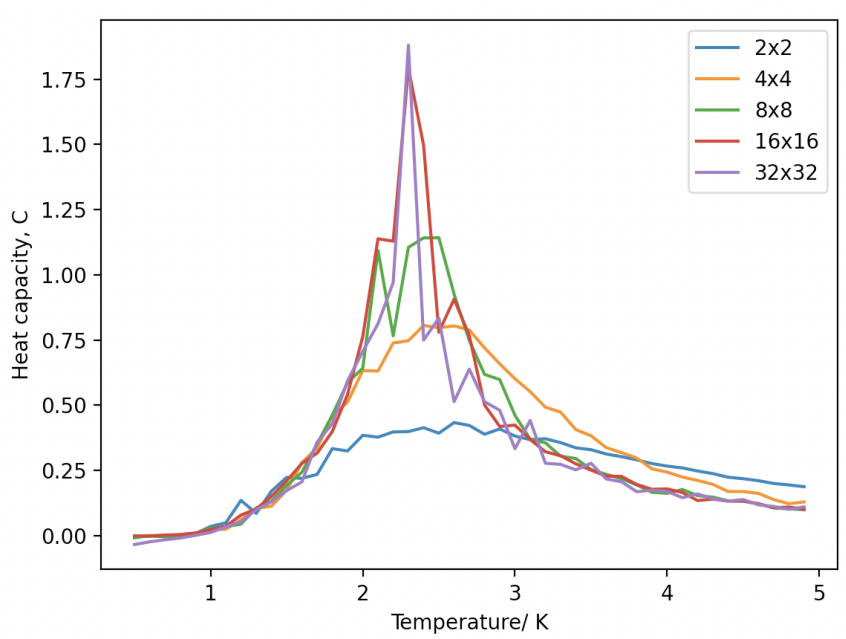


Figure 10: Heat capacity plotted for a range of grid sizes.

The reason the heat capacity of an infinite lattice diverges at its Curie temperature is due to the divergence of the correlation length (Equation 35). This divergence results in strongly correlated spins over large distances, increasing the energy required to change the system's temperature, the heat capacity, as flipping one spin requires flipping many other correlated spins.[9] As a result, we get Equation 36 modelling heat capacity's divergence.

$$\xi \sim \frac{1}{|(T - Tc)|^{\nu}} \tag{35}$$

$$C \sim \frac{1}{|(T - Tc)|^{\alpha}} \tag{36}$$

9 Task 8

9.1 Task 8a

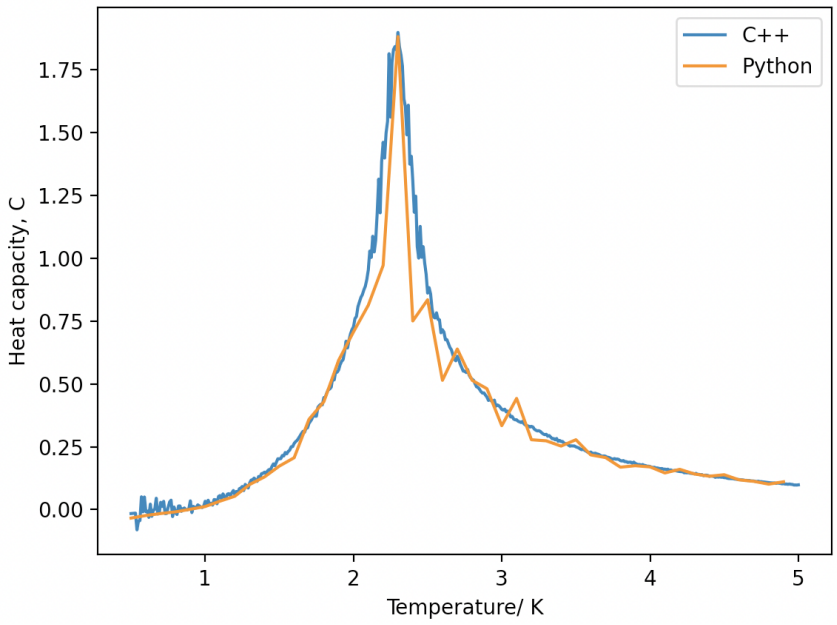


Figure 11: Heat capacity plots comparing python and C++ data for the 32x32 lattice.

Due to the speed of C++ programs, is it possible to run many more steps in the same amount of time as for a python program. Figure 11 shows the comparison between running the program in C++ and python for the heat capacity and Figure 12 for the energy and magnetisation. The plots made from C++ data have more data points as they record the temperature every 0.02 K instead of 0.1 K in python. This produces a smoother plot for heat capacity, but is more important in the magnetisation plot. The C++ data for the magnetisation plot in Figure 12 is recorded on a small enough interval to observe the fluctuations in magnetisation around the T_C, which our larger temperature interval in python completely misses with its more smooth curve. The energy plots are effectively identical due to the less extreme fluctuations in energy compared to magnetisation. This is due to the fact that a single energy can have many microstates with the same magnetisation, yielding a smooth energy curve but a fluctuating magnetisation curve.

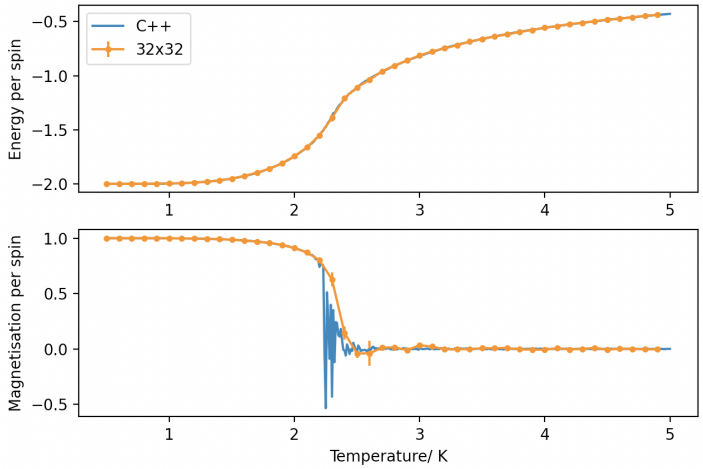


Figure 12: Energy and magnetisation plots comparing C++ and python data for the 32x32 lattice. The legend applies to both plots.

9.2 Task 8b

Due to the noise of the data, it is best if we fit the curve if we want to use it to estimate the Curie temperature. Figure 13 shows the entire curve fitted to a polynomial of order 5. The aim of this fit was to capture the general shape of the curve. Using a higher order polynomial, the curve can be fit almost perfectly, however, this would be over fitting as it would be fitting all the random fluctuations and not actually the general shape of the curve.

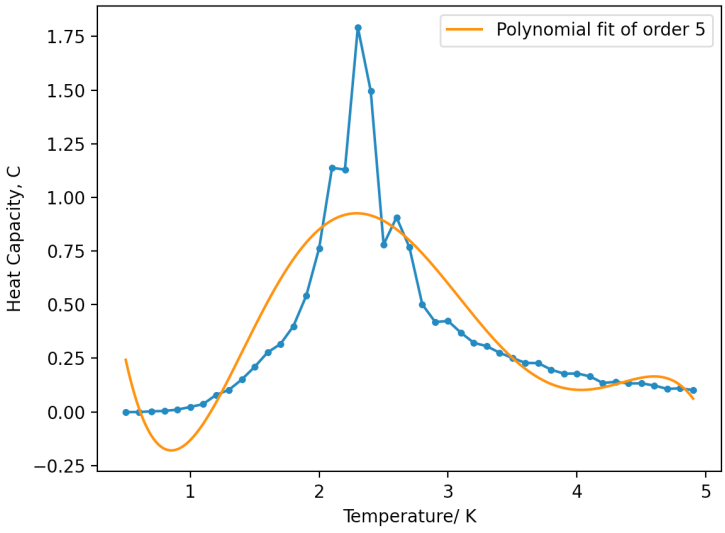


Figure 13: Fit of polynomial order 5 for heat capacity vs T curve of a 16x16 grid. There are few points in the sharp peak meaning they do not contribute much weighting to the fit so the fit curve is below the peak.

9.3 Task 8c

Fitting the entire curve is less important when trying to estimate the Curie temperature. The curve actually needs to be fit over the temperature range around the peak, Figure 14. Fitting the peak gives us a more reliable value of T_C instead of just using the point at the peak as the T_C. After fitting the peak in Figure 14 it was found $T_C = 2.318 \pm 0.022K$ for the 16x16 grid.

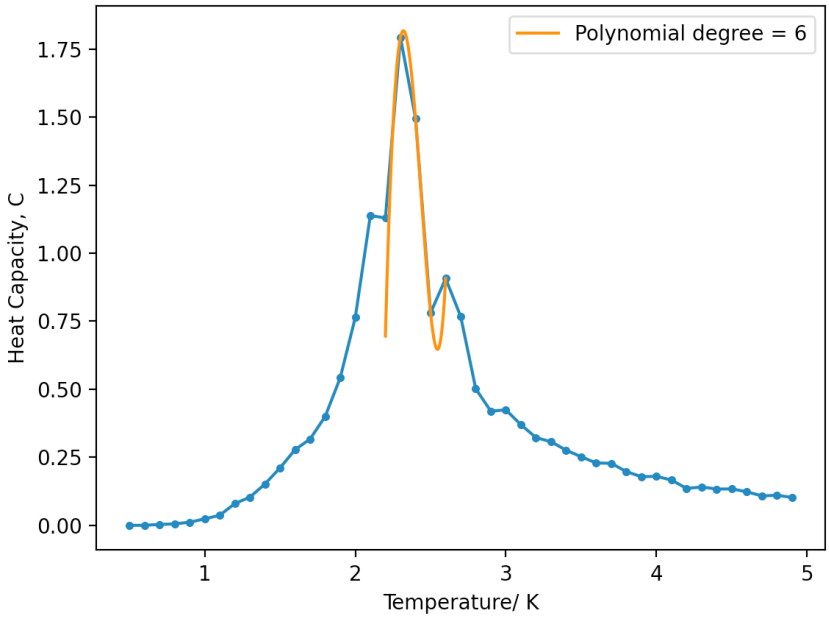


Figure 14: Fit of polynomial order 6 of the T range around the peak for a 16x16 grid.

The following code was used to fit the peak:

```
data = np.loadtxt("16x16.dat")
T = data[:,0]
C = (data[:,2] - data[:,1]**2)/(T**2*16**2)
varE_2 = data[:,2] - data[:,1]**2
Tmin = 2.2
Tmax = 2.6

selection = np.logical_and(T > Tmin, T < Tmax) #choose only those rows
# where both conditions are true
peak_T_values = T[selection]
peak_C_values = C[selection]

peak_T_range = np.linspace(Tmin, Tmax, 1000)

fit = np.polyfit(peak_T_values, peak_C_values, 6)
fitted_C_values = np.polyval(fit, peak_T_range)

Cmax = np.max(fitted_C_values)
T_c = peak_T_range[fitted_C_values == Cmax]
```

9.4 Task 8d

$T_{C,\infty}$ is the curie temperature of an infinite lattice, which is related to the curie temperature of an L sized lattice, $T_{C,L}$, by the following formula:

$$T_{C,L} = \frac{A}{L} + T_{C,\infty} \tag{37}$$

Where A is a constant. This means that by plotting $T_{C,L}$ against $\frac{1}{L}$, the y-intercept will be $T_{C,\infty}$. The code used in Task 8c was applied to lattice sizes 2x2 4x4 8x8 16x16 and 32x32 in order to find the

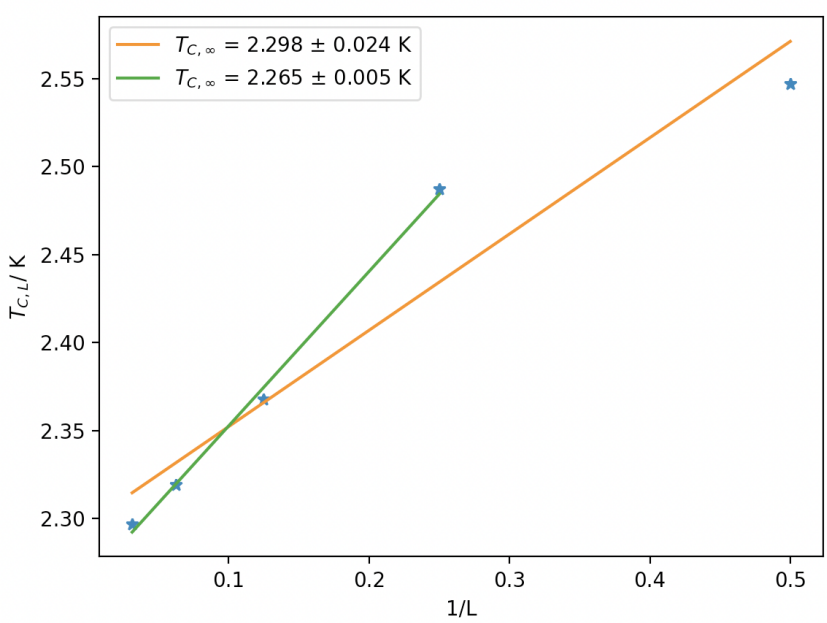


Figure 15: Graph fitted including all points and also excluding the 2x2 grid for higher accuracy.

Curie temperature for each grid size, $T_{C,L}$. A plot of $T_{C,L}$ against $\frac{1}{L}$ is seen in Figure 15. Two different fits are shown: one fit for all the grid sizes recorded and one fit excluding the smallest grid size. The gridsize excluded was the 2x2 grid, as in this grid the spins can effect themselves as they are so close, yielding inaccurate results and being a major source of error. The literature value for the theoretical exact Curie temperature for the infinite 2D Ising lattice is 2.269 K.[10] Including the 2x2 grid in the fit yields $T_{C,\infty} = 2.298 \pm 0.024K$, which is very close but not within error of the literature value. Excluding the 2x2 grid yields $T_{C,\infty} = 2.265 \pm 0.005K$, which is within error with the literature value, showing the inaccuracy added by the 2x2 grid. These results are extremely good, and unexpected that a model as simple as the Ising model would produce such accurate results. In order to reduce error, T_C can be found for larger lattice sizes and used for a better fit. One of the main sources of error is Equation 37 as it involves a first order taylor expansion about the thermodynamic limit, where L $\rightarrow \infty$ (1/L $\rightarrow$ 0), assuming this relation to be linear. Therefore Equation 37 is actually only valid for large L, justifying excluding the 2x2 grid in the fit. In theory, we would have grid sizes 64, 128 and 256 and exclude 4x4 and 8x8 in the fits to get a more accurate value of $T_{C,\infty}$ if it was not so computationally expensive in python, or even to use a higher order approximation if it was not too complex.

10 Conclusion

In conclusion, the Ising model has yielded surprisingly accurate results for predicting the Curie temperature of an infinite sized lattice, however, using larger grid sizes such as 64x64 and 128x128 would increase the accuracy further, but at the cost of computation time. One problem that was encountered was systems getting stuck in metastable states (local minima), such as in Figure 16. The temperature was too low to overcome the energy barrier to exit the state and reach true equilibrium in a sensible number of steps. In this case actually, you can see that a interface has formed, which would also require energy to overcome, making this potential well hard to escape (but the interface is flat and not curved, suggesting another possible class of even more stubborn metastable states with curved interfaces that we did not observe).
In the future it would be interesting to investigate the Ising model using the e.g. the Wolff algorithm instead of the Metropolis algorithm. The Wolff algorithm is a cluster algorithm meaning it flips a cluster of spins at once instead of individual spins. This makes it more likely for a system to escape a local minimum in a sensible number of steps and will make the results more reliable and less effected by the random starting configuration of spins.[11]

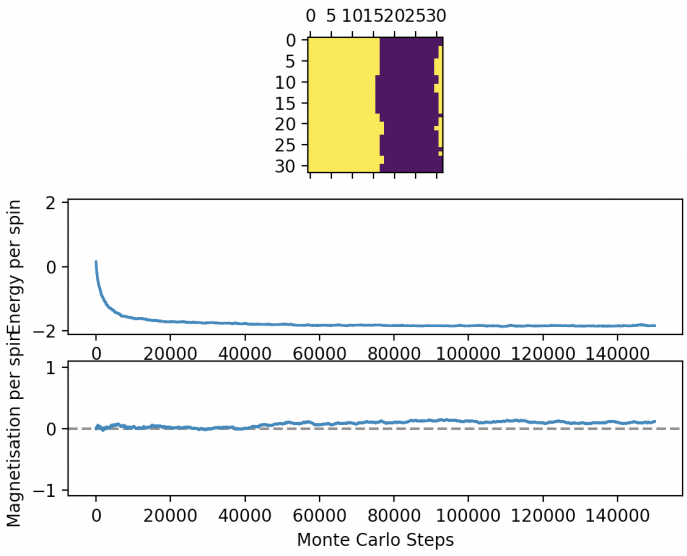


Figure 16: Metastable state of 32x32 lattice.

References

[1] S. P. Singh, in *Solid State Physics-Metastable, Spintronics Materials and Mechanics of Deformable Bodies-Recent Progress*, IntechOpen, 2020.

[2] J. Jar, *The Ising Model*, 2020, https://stanford.edu/~jeffjar/statmech/intro4.html, [Accessed on 03/21/2023].

[3]

[4] L. Onsager, *Phys. Rev.*, 1944, **65**, 117–149.

[5] F. Bresme, *Introduction to the Ising model*, 2020, https://wiki.ch.ic.ac.uk/wiki/index.php?title=Third_year_CMP_compulsory_experiment/Introduction_to_the_Ising_model, [Accessed on 03/21/2023].

[6] Y. Xie, J. Fan, L. Xu, X. Zhang, R. Xu, Y. Zhu, R. Tang, C. Wang, C. Ma, L. Pi *et al.*, *Physics Letters A*, 2019, **383**, 125843.

[7] F. Foo, *Why are Numpy arrays so fast?*, 2011, https://stackoverflow.com/questions/8385602/why-are-numpy-arrays-so-fast, [Accessed on 03/21/2023].

[8] F. Krauss and D. Maitre, *Ising Model - Phase Transitions*, 2017, https://www.ippp.dur.ac.uk/~krauss/Lectures/NumericalMethods/PhaseTransitions/Lecture/pt2.html, [Accessed on 03/21/2023].

[9] J. M. Kim, *Journal of the Korean Physical Society*, 2022, **81**, 602–607.

[10] X. Zhao, *Determining the critical temperature of Ferromagnetism by using Monte Carlo Algorithm*, 2018, https://www.s.u-tokyo.ac.jp/en/utrip/archive/2019/pdf/2_03.pdf, [Accessed on 03/21/2023].

[11] E. Luijten, *Computer Simulations in Condensed Matter Systems: From Materials to Chemical Biology Volume 1*, 2006, 13–38.

11 Code

11.1 IsingLattice.py

```
import numpy as np

class IsingLattice:

    E = 0.0
    E2 = 0.0
    M = 0.0
    M2 = 0.0

    n_cycles = 0

    def __init__(self, n_rows, n_cols):
        self.n_rows = n_rows
        self.n_cols = n_cols
        self.lattice = np.random.choice([-1,1], size=(n_rows, n_cols))
        #self.N = n_rows*n_cols
        self.J = 1

        # def f(x, a, b, c):
        #     return a + b*np.exp(c*x)

        def f(x, a, b):
            return a*np.exp(b*x)

        #self.cutoff = f(n_rows, a = -2.00899300e+03, b = 8.71557693e+02, c = 1.59759578
        self.cutoff = f(n_rows, a = 10**(-0.52), b=4.11)


    def energy(self):
        """
        Return the total energy of the current lattice configuration.
        """
        # shift the lattice by 1 in each direction
        # Just 2 directions to avoid double counting
        R = np.roll(self.lattice, shift=-1, axis=0)
        UP = np.roll(self.lattice, shift=-1, axis=1)

        # Compute the sum of neighbouring spins
        sum_sisj = np.sum(self.lattice * (R + UP), axis=(0, 1))

        # Compute and return the energy
        energy = -self.J * sum_sisj
        return energy


    def magnetisation(self):
        "Return the total magnetisation of the current lattice configuration."
        magnetisation = np.sum(self.lattice)

        return magnetisation


    def montecarlostep(self, T):
        self.n_cycles += 1
        #can do this in 1-2 if statements
        #generating a random number lets you test the probability distribution for one
```

```
        # complete this function so that it performs a single Monte Carlo step
        energy = self.energy()

        #the following two lines will select the coordinates of the random spin for you
        random_i = np.random.choice(range(0, self.n_rows))
        random_j = np.random.choice(range(0, self.n_cols))

        # flip the spin
        self.lattice[random_i, random_j] = -1*self.lattice[random_i, random_j]

        # calc energy of new config
        new_energy = self.energy()
        E_diff = new_energy - energy

        #the following line will choose a random number in the rang e[0,1) for you
        random_number = np.random.random()

        botlzmann = np.exp(-E_diff/(T)) #*1.38e10-23

        if (E_diff > 0 and random_number > botlzmann):
            # reject
            self.lattice[random_i, random_j] = -1*self.lattice[random_i, random_j]

        else:
            # accept the copy and return the new config and magnetisation
            energy = new_energy

        M = self.magnetisation()

        if self.n_cycles > self.cutoff:
            self.E += energy
            self.E2 += energy**2
            self.M += M
            self.M2 += M**2
        else:
            pass

        return energy, M


    def statistics(self):
        #complete this function so that it calculates the correct values for the average
        #Z = (2*np.cosh(self.J/1))**(self.N)
        N = self.n_cycles
        cutoff = self.cutoff
        if N > cutoff:
            avg_E = self.E/(N-cutoff)
            avg_E2 = self.E2/(N-cutoff)
            avg_M = self.M/(N-cutoff)
            avg_M2 = self.M2/(N-cutoff)

        return avg_E, avg_E2, avg_M, avg_M2, N
```