FYS4150 - Computational Physics
Project 4

Alex Ho

November 13, 2016

Abstract

We will take a look at the Ising model and implement the Metropolis method to solve the Ising model numerically. With this, we observe the behaviour of the energy and magnetization for different lattice sizes. We also will also study the behaviour of the Ising model near the critical temperature.

Contents

1 Introduction **3**

2 Method **3**
2.1 The Ising model 3
2.2 Metropolis algorithm and computation time 5
2.3 Parallel computation 7
2.4 Simple 2 × 2 lattice 7
2.5 Studies of phase transition 12

3 Implementation **13**

4 Results **14**
4.1 Comparing with the analytical solution 14
4.2 Lattice with L = 20 16
4.3 Probability distribution 26
4.4 Phase transitions 28

5 Conclusion **28**

6 Appendix **29**

1 Introduction

In fields like thermal dynamics, one will study the phenomena called a *phase transition*. Not only do we study it, but we experience this phenomena on a day-by-day basis. An example of a phase transition is when water turns to steam when we boil the water. More general, a phase transition is when matter changes it's form, either from gas to liquid, or liquid to solid (and vice versa).

We will in this project use a very popular model, the Ising model, to look at the statistical properties of phase transitions analytically. We will first consider a small 2×2 lattice and see how the Metropolis algorithm compares to the analytical solutions for this system. After that, we will see how this system evolves with time as we increase the lattice size. Finally, we will take a look at phase transitions near the critical temperature for very, very large lattices.

All relevant files can be found in this GitHub page:

`https://github.com/AHo94/FYS3150_Projects/tree/master/Project4`

2 Method

2.1 The Ising model

The Ising model is a mathematical model for magnetic systems in statistical mechanics. We will in this project consider phase transitions at finite temperatures. The energy, for a specific microstate i, is expressed as

$$E_i = -J \sum_{<kl>}^{N} s_k s_l - \mathcal{B} \sum_{k}^{N} s_k \tag{1}$$

With the spins s_k which can take values $s_k = \pm 1$. N is the total number of spins in the system and J is a coupling constant, which we will assume to

be $J > 0$. The symbol $< kl >$ indicates that we only sum over the nearest neighbouring spin. That is, we only the spin s_k with the closest spin s_l. $\mathcal{B}$ is an external magnetic field that is interacting with the magnetic moment between neighbouring spins. For simplicity of this project, we will set $\mathcal{B} = 0$.

We also have the magnetization (or magnetic moment) defined as

$$\mathcal{M} = \sum_{j=1}^{N} s_j \tag{2}$$

What we are interested in is the expectation values of the energy $\langle E \rangle$ and the magnetization $\langle \mathcal{M} \rangle$. In order to do that, we will need a probability distribution

$$P_i(\beta) = \frac{e^{-\beta E_i}}{Z} \tag{3}$$

where $\beta = 1/k_b T$, the inverse temperature with k_b as the Boltzmann constant. The partition function Z is given as

$$Z = \sum_{i=1}^{M} e^{-\beta E_i} \tag{4}$$

which sums over all possible microstates M. With these, we can calculate the expectation value of an arbitrary variable x

$$\langle x \rangle = \frac{1}{Z} \sum_i x_i e^{-\beta E_i}$$

$$\langle x^2 \rangle = \frac{1}{Z} \sum_i x_i^2 e^{-\beta E_i}$$

The variance for this variable is then

$$\sigma_x^2 = \langle x^2 \rangle - \langle x \rangle^2$$

We can now use these properties to calculate the heat capacity C_v and susceptibility χ, which are given as

$$C_v = \frac{\sigma_E^2}{k_B T^2}$$

$$\chi = \frac{\sigma_M^2}{k_B T}$$

2.2 Metropolis algorithm and computation time

Monte Carlo methods are widely used to solve problems in different fields in science. The idea behind Monte Carlo simulation is do statistical simulations by drawing a sequence of random numbers In this project, we will use the Monte Carlo method known as the Metropolis algorithm to solve the Ising model numerically. The algorithm are as follows: for each Monte Carlo cycle we do:

- **1)** We start with an arbitrary microstate for the given lattice. We use that microstate to calculate the energy E_b
- **2)** Choose a random spin in the microstate and flip it (e.g, from $\uparrow$ to $\downarrow$). Now we use that state to calculate the energy E_t.
- **3)** Calculate the energy difference $\Delta E = E_t - E_b$.
- **4)** If $\Delta E <= 0$, we accept the new spin configuration and use the energy E_t. What this means is that the energy has been lowered and we are in a lower state.
- **5)** If $\Delta E > 0$, pick a random number r that is given from a uniform distributor in the range $r \in [0, 1]$. Now we have to consider these two cases
- **5a)** If $r <= e^{-\beta \Delta E}$, we accept the new spin configuration and use the energy E_t.
- **5b)** If $r > e^{-\beta \Delta E}$, we reject this new configuration and flip the chosen spin back to it's original state and use the energy E_b.
- **6)** Once that is done, we use the given microstate (depending on the cases given above) to calculate the magnetic moment M_i.
- **7)** Finally, we add the calculated energy and magnetic moment to E_{sum} and M_{sum} respectively. These will the one sample of the Monte Carlo cycle.

We repeat this process N_{mc} times (with N_{mc} as the number of Monte Carlo cycles) and multiple samplings which will be summed into E_{sum} and M_{sum}. Once that is done, we get the mean energies as

$$\langle E \rangle = \frac{1}{N_{mc}} E_{sum} = \frac{1}{N_{mc}} \sum_i E_i$$

With E_i is energy sample of the i'th Monte Carlo cycle. Similar expressions are used for the magnetic moment and their squares.

Since we are working with probabilities, we will have to run this algorithm many, many times. Take a dice roll as an example. We know that the probability of getting the number 3 is 1/6. If we were to throw the dice 10 times, the probability of getting 3 will most likely not be 1/6 any more. One have to throw the die many many times to get the expected probability. The same idea is applied for Monte Carlo methods, as one has to do many samplings to get a reasonable result. This is also why Monte Carlo simulations are known to be greedy simulations (in terms of computation time).

Because of this, doing Monte Carlo simulation will take a lot of computation time. Since we have to change all the possible spins L, the number of floating point operations our algorithm will be roughly L^2. This is not including other intermediate calculations within the algorithm. When we consider $L = 2$, the computation time will be incredibly short. However, as we increase the total number of spins to $L = 20, 40, 60, 100, 140$, the computation time will increase by roughly $400, 1600, 3600, 10000$ and 19600 times respectively.

Note that this is only for one Monte Carlo cycle. As mentioned earlier, we will have to do many Monte Carlo cycles to obtain a reasonable result. This will increase the computation time to roughly $N_{mc}L^2$. On top of that, we will have to do these simulations for different temperature T, so the simulations will increase yet again to roughly $N_T N_{mc} L^2$, where N_T is the number of temperatures we would like to have. To save time, we will have to do Parallel computation, which will explained in the next part.

2.3 Parallel computation

As mentioned in the previous part, there will be very long computation time once we increase the lattice size. In order to save time, we will have to use multiple cores, that should be provided in a modern computer or laptop, to do our simulation. The idea is to let the core work in parallel with each other, that is, to split the computation work among different cores, which will speed up our process quite significantly.

We will use openMPI to do parallel computing. One has to first specify the number of cores the computer has available, and openMPI will automatically split these cores to different nodes (or ranks). Each node will run the program separately, but we can specify certain initial numbers, or intervals, for each node. One can, for example, tell each node to do the computation for different temperatures. One can also specify which interval each nodes computes. For example, node 0 can compute the Monte Carlo cycles in the range $N_{mc} \in [0, 100]$, while node 1 computes in the interval $N_{cm} \in [101, 200]$. If we were to do the latter option, we will have to keep in mind that the nodes are working separately, and the computed values will therefore be unknown to each other. We will have to merge the values together to one main node (or the master node), which can be done by using the command `MPI_Reduce`.

The computation time will decrease proportionally with the number of cores we use for the simulation. The computation time will be roughly $(N_T N_{mc} L^2)/N_c$, where N_c is the number of cores used in the computation.

2.4 Simple 2 $\times$ 2 lattice

We will first consider a 2×2 lattice, find the analytical expression for partition function Z and find the corresponding expectation value of energy E, mean magnetization $|M|$, specific heat C_V and susceptibility ξ as functions of temperature T. We also consider periodic boundary condition for this lattice. Once we have the analytical expressions, we will compare them to the numerical values from the Metropolis algorithm.

Combinations of	(s_1, s_2, s_3, s_4)	$s_j = \{\uparrow, \downarrow\} = \{1, -1\}$	
$(\uparrow, \uparrow, \uparrow, \uparrow)$	$(\uparrow, \uparrow, \uparrow, \downarrow)$	$(\uparrow, \uparrow, \downarrow, \uparrow)$	$(\uparrow, \downarrow, \uparrow, \uparrow)$
$(\downarrow, \uparrow, \uparrow, \uparrow)$	$(\uparrow, \uparrow, \downarrow, \downarrow)$	$(\uparrow, \downarrow, \uparrow, \downarrow)$	$(\downarrow, \uparrow, \uparrow, \downarrow)$
$(\downarrow, \uparrow, \downarrow, \uparrow)$	$(\downarrow, \downarrow, \uparrow, \uparrow)$	$(\uparrow, \downarrow, \downarrow, \uparrow)$	$(\uparrow, \downarrow, \downarrow, \downarrow)$
$(\downarrow, \uparrow, \downarrow, \downarrow)$	$(\downarrow, \downarrow, \uparrow, \downarrow)$	$(\downarrow, \downarrow, \downarrow, \uparrow)$	$(\downarrow, \downarrow, \downarrow, \downarrow)$

Table 1: All the microstates possible.

For this system, we will assume that every spin has two directions, i.e. our states can be either be in spin up state or spin down state (shorthand notation as $\uparrow$ or $\downarrow$ respectively).

The energy of the Ising model, without an external magnetic field $\mathcal{B}$, is given by

$$E_i = -J \sum_{<kl>}^{N} s_k s_l$$

Where $J > 0$ is a coupling constant and N is the total number of spins. The symbol $< kl >$ indicates that we only sum over the neighbours only. The values $s_k = \pm 1$ depends on which state it is in. We let $s_\downarrow = -1$ and $s_\uparrow = 1$. We also have the magnetic moment is given as

$$M_i = \sum_{<k>}^{N} s_k$$

Since we have a $2 \times 2 = 4$ lattice, and we have two spin directions, then the number of microstate (or configuration) is $2^4 = 16$. What this means is that our we can have 16 different energies, as well as 16 different magnetic moment, for each respective microstate. Table 1 shows all the possible microstates.

Figure 1 shows a 2×2 lattice. We see that the point s_1 has s_2 and s_3 as the closest neighbours. Since we are considering periodic boundary conditions, then s_1 will connect to s_2 and s_3 twice. The energy term will then give the term $2(s_1 s_2 + s_2 s_3)$ for the point s_1. It does not include s_4 as it is not the closest neighbour to s_1.

We can then continue to add more terms using the three other points, but we need to be careful to not include the connections of the points we previously

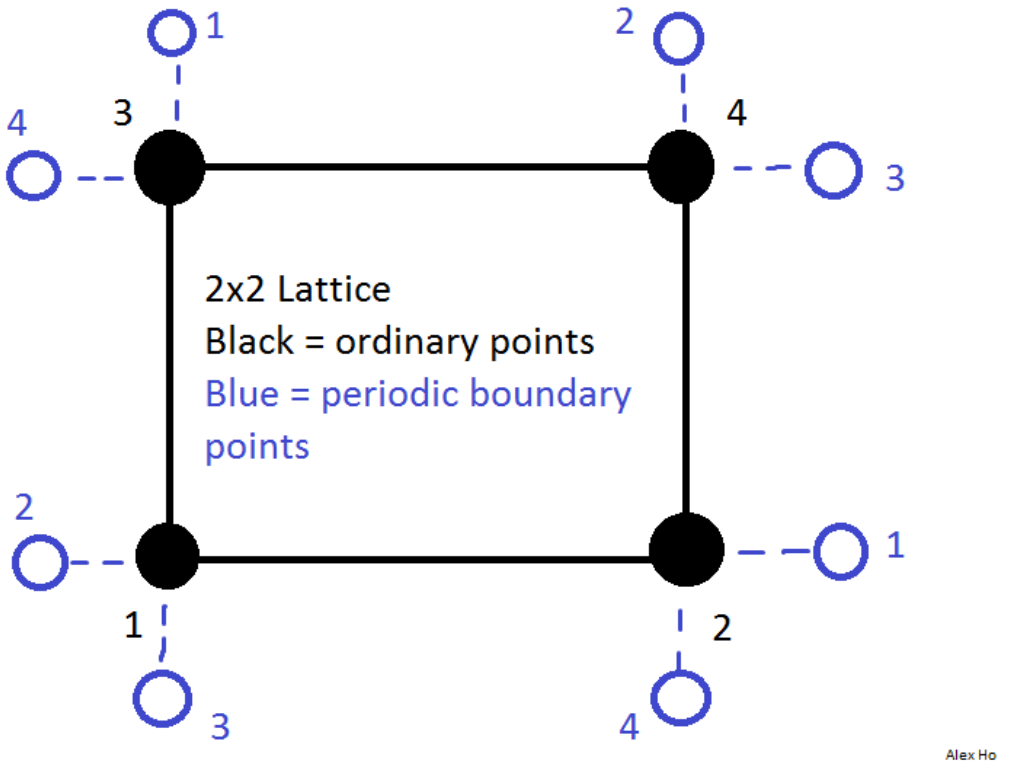

Figure 1: An illustration of the 2×2 lattice. The black points corresponds to the ordinary points s_1, s_2, s_3, s_4 (as point 1, 2, 3, 4 in the figure respectively). The blue points corresponds periodic boundary points.

have considered, which is to prevent double counting. Doing this, the energy for each microstate i will be

$$E_i = -2J \sum_{s_1=\pm 1} \sum_{s_2=\pm 1} \sum_{s_3=\pm 1} \sum_{s_4=\pm 1} (s_1 s_2 + s_1 s_3 + s_2 s_4 + s_3 s_4) \tag{5}$$

Similarly for the magnetic moment we get when we sum over all microstates

$$M_i = \sum_{s_1=\pm 1} \sum_{s_2=\pm 1} \sum_{s_3=\pm 1} \sum_{s_4=\pm 1} (s_1 + s_2 + s_3 + s_4) \tag{6}$$

Let us now determine both the energies and magnetic moments for all microstates. Using table 1, we can determine equation (5) and (6) to their respective microstate. Table 2 and 3 shows the energies and magnetic momenta (using the same combinations in table 1) respectively.

$E_i =$			
-8J	0	0	0
0	8J	0	0
0	8J	0	0
0	0	0	-8J

Table 2: Energies for each respective microstate.

$M_i =$			
4	2	2	2
2	0	0	0
0	0	0	-2
-2	-2	-2	-4

Table 3: Magnetic moments for each respective microstate.

Now that we have the energies of each microstate, we can find an analytical expression for the partition function Z given in 4. Using the energies given in table 2, the partition function becomes

$$Z = 2e^{8\beta J} + 2e^{-8\beta J} + 12e^{0} = 4\cosh(8\beta J) + 12$$

With the partition function, we can calculate the expectation value of the energy

$$\langle E \rangle = \frac{1}{Z} \sum_i E_i e^{-\beta E_i}$$

Summing over all states i, with the given energies in table 2, we get

$$\begin{aligned} \langle E \rangle &= \frac{1}{Z} \left(2(8J)e^{-8\beta J} + 2(-8J)e^{8\beta J} + 12 \times 0 \times e^{0} \right) \\ &= \frac{-32J \sinh(8\beta J)}{4\cosh(8\beta J) + 12} \\ &= \frac{-8J \sinh(8\beta J)}{\cosh(8\beta J) + 3} \end{aligned}$$

The expectation of the energy squared is then

$$\begin{aligned}\langle E^2\rangle &= \frac{1}{Z}\sum_i E_i^2 e^{-\beta E_i} \\ &= \frac{1}{Z}\left(2(8J)^2 e^{-8\beta J} + 2(-8J)^2 e^{8\beta J} + 12\times(0)^2 e^0\right) \\ &= \frac{4(8J)^2\cosh(8\beta J)}{4\cosh(8\beta J)+12} \\ &= \frac{64J^2\cosh(8\beta J)}{\cosh(8\beta J)+3}\end{aligned}$$

The standard deviation of the energy then becomes

$$\begin{aligned}\sigma_E^2 &= \langle E^2\rangle - \langle E\rangle^2 \\ &= \frac{64J^2\cosh(8\beta J)}{\cosh(8\beta J)+3} - \left(\frac{-8J\sinh(8\beta J)}{\cosh(8\beta J)+3}\right)^2 \\ &= \frac{64J^2}{\cosh(8\beta J)+3}\left(\cosh(8\beta J) - \frac{\sinh^2(8\beta J)}{\cosh(8\beta J)+3}\right) \\ &= \frac{64J^2}{\cosh(8\beta J)+3}\left(\frac{\cosh^2(8\beta J)+3\cosh(8\beta J)}{\cosh(8\beta J)+3} - \frac{\sinh^2(8\beta J)}{\cosh(8\beta J)+3}\right) \\ &= \frac{64J^2}{\cosh(8\beta J)+3}\left(\frac{1+3\cosh(8\beta J)}{\cosh(8\beta J)+3}\right) \\ &= \frac{64J^2}{(\cosh(8\beta J)+3)^2}(4+\cosh(8\beta J))\end{aligned}$$

Dividing by $k_B T^2$ gives us the specific heat C_V

$$C_V = \frac{1}{k_B T^2}\left(\langle E^2\rangle - \langle E\rangle^2\right) = \frac{64J^2}{k_B T}\frac{(1+3\cosh(8\beta J))}{(\cosh(8\beta J)+3)^2} \tag{7}$$

We can do similar calculations to obtain the mean magnetization (or the mean absolute value of the magnetic moment), which then gives us

$$\langle |M|\rangle = \frac{1}{Z}\sum_i |M_i| e^{-\beta E_i} = \frac{2(e^{8\beta J}+2)}{\cosh(8\beta J)+3}$$

$$\langle M^2\rangle = \frac{1}{Z}\sum_i M_i^2 e^{-\beta E_i} = \frac{8(e^{8\beta J}+1)}{\cosh(8\beta J)+3}$$

Which we can use to calculate the susceptibility χ

$$\chi = \frac{1}{k_B T}\left(\langle M^2\rangle - \langle |M|\rangle^2\right) = \frac{8}{k_B T}\frac{(e^{8\beta J} + \cosh(8\beta J) + \frac{3}{2}))}{(\cosh(8\beta J) + 3)^2} \tag{8}$$

Now that we have the analytical expressions, we will later compare it to the numerical values (for a given temperature) from the Metropolis algorithm.

2.5 Studies of phase transition

Once we have properly tested the Ising model by using the Metropolis algorithm, we will study the behaviour of the Ising model near the critical temperature T_C.

Through so-called finite size scaling relations, we can relate the behaviour of finite sized lattices with the results of an infinitely large lattice. The critical temperature then scales as

$$T_C(L) - T_C(L = \infty) = aL^{-1/\nu} \tag{9}$$

Where a is a constant, and ν is defined from

$$\xi(T)\ |T_C - T|^{-\nu}$$

For our case, we will set $\nu = 1$. This is to compare the exact result of the critical temperature after Lars Onsager, which is.

$$\frac{kT_C}{J} = \frac{1}{\ln(1+\sqrt{2})} \approx 2.269$$

With $\nu = 1$. First, we will have to find what this constant a is. By taking the difference of equation 9 for two different total spins, we have

$$\begin{aligned} aL_i^{-1} - aL_j^{-1} &= [T_C(L_i) - T_C(L = \infty)] - [T_C(L_j) - T_C(L = \infty)] \\ &= T_C(L_i) - T_C(L_j) \\ \implies a &= \frac{T_C(L_i) - T_C(L_j)}{L_i^{-1} - L_j^{-1}} \end{aligned}$$

The critical temperature for an infinitely large lattice, for a given total spin L_i, is then

$$\begin{aligned} T_C(L =_i \infty) &= T_C(L_i) - L_i^{-1}\frac{T_C(L_i) - T_C(L_j)}{L_i^{-1} - L_j^{-1}} \\ &= T_C(L_i) - \frac{T_C(L_i) - T_C(L_j)}{1 - \left(\frac{L_i}{L_j}\right)} \end{aligned}$$

Since we have multiple total spins, we will have to sum over all the other total spins, and then average them out. That is

$$T_C(L_i = \infty) = \frac{1}{N_L}\sum_j \left(T_C(L_i) - \frac{T_C(L_i) - T_C(L_j)}{1 - \left(\frac{L_i}{L_j}\right)} \right)$$

Where N_L is the number of all lattice sizes considered. We will for this part let the total spins be $L = 40, 60, 100$ and 140, which means that $N_L = 4$. We will also consider the temperature range $T \in [2.0, 2.3]$.

3 Implementation

All programming are done in C++ and data will be saved in a text file. The plotting part will be done in Python.

The main function in the C++ program is split into two parts by an if test. The if test will see if there is any arguments given in the command line. If there is none, the program will compute the simple lattice model for $L = 2$ and $L = 20$. If the there are arguments in the command line, the program will run the program for large lattices. The program will here do parallel computation, where I have decided to separate the Monte Carlo cycle interval for each node.

There are also various of functions that is used to do all the calculations. Some of the functions have relatively similar names. One example is the `write_file` functions. I have made two (or three, if you include the function

that initializes the output file) of these functions. Both creates differently formatted output files, which is done because it will be easier to read the required data for the different tasks.

Like my previous projects, I will plot all the data, made from the C++ program, in Python. The Python script simply reads the output file, saves the required data and then plots them.

4 Results

4.1 Comparing with the analytical solution

We will now compare the numerical values for C_v and χ with their respective analytical values given in equation 7 and 8. For this case, we will use the temperature $T = 1.0$ in units of kT/J. The coupling constant J will be set to $J = 1$ throughout the whole project. Running this in C++, with 10^6 Monte Carlo cycles we obtain the following output (Note that the numerical values has not been divided by the number of spins):

```
Number of Monte Carlo cycles = 1000000
Analytic C_v = 0.128329, Numerical C_v = 0.128445
Analytic Chi = 0.016043, Numerical Chi = 0.0162828
```

As we can see, the values are almost identical. The numerical values may change a little bit when we run this multiple times, but still remains fairly close to the analytical values. An example of this can be found in the output below

```
Running multiple times, using MC_cycles = 1000000
Analytic C_v = 0.128329, Numerical C_v = 0.123027
Analytic Chi = 0.016043, Numerical Chi = 0.0156212

Analytic C_v = 0.128329, Numerical C_v = 0.127936
Analytic Chi = 0.016043, Numerical Chi = 0.0154724
```

```
Analytic C_v = 0.128329, Numerical C_v = 0.130357
Analytic Chi = 0.016043, Numerical Chi = 0.0164741

Analytic C_v = 0.128329, Numerical C_v = 0.125385
Analytic Chi = 0.016043, Numerical Chi = 0.0152174

Analytic C_v = 0.128329, Numerical C_v = 0.126915
Analytic Chi = 0.016043, Numerical Chi = 0.0156962

Analytic C_v = 0.128329, Numerical C_v = 0.124365
Analytic Chi = 0.016043, Numerical Chi = 0.0159204
```

Using 10^5 Monte Carlo cycles can also give a good numerical result, but it is not as consistent compared to 10^6 Monte Carlo cycles. What that means is that the difference between the analytical and numerical values (for both C_v and χ) may be larger for a lower number of Monte Carlo cycles. The output below shows an example with $N_{mc} = 10^5$. Notice how, for example, the numerical values of C_v can be as low as 0.112442 and as high as 0.160237.

```
Running multiple times, using MC_cycles = 100000
Analytic C_v = 0.128329, Numerical C_v = 0.125194
Analytic Chi = 0.016043, Numerical Chi = 0.0163315

Analytic C_v = 0.128329, Numerical C_v = 0.112442
Analytic Chi = 0.016043, Numerical Chi = 0.0130196

Analytic C_v = 0.128329, Numerical C_v = 0.123282
Analytic Chi = 0.016043, Numerical Chi = 0.0152538

Analytic C_v = 0.128329, Numerical C_v = 0.137304
Analytic Chi = 0.016043, Numerical Chi = 0.0180454

Analytic C_v = 0.128329, Numerical C_v = 0.125832
Analytic Chi = 0.016043, Numerical Chi = 0.0161317

Analytic C_v = 0.128329, Numerical C_v = 0.160237
Analytic Chi = 0.016043, Numerical Chi = 0.0194764
```

4.2 Lattice with L = 20

We will now increase our lattice to a 20×20 lattice. For this task, we will have a look at how the mean energy and mean magnetization (absolute value) will reach an equilibrium as a function of the number of Monte Carlo cycles. Monte Carlo cycles will represent the time for this case. This will be done for temperatures at $T = 1.0$ and $T = 2.4$. We will also consider two different cases. One where all the spins in the system is randomly set and one where all spins are pointing up.

Let us first consider the case where the initial spin state is set at random. Figure 2 and 3 are plots of the expectation values of energy and magnetization respectively, as a function of time (the number of Monte Carlo cycles) for $T = 1.0$. The x axis is plotted in logarithmic scales to better show how the expectation values stabilizes over time. Non-logarithmic x-axis of these plots can be found in the appendix (also found in my GitHub page). We see that both parameters stabilizes fairly quickly, even though we start in a random spin state.

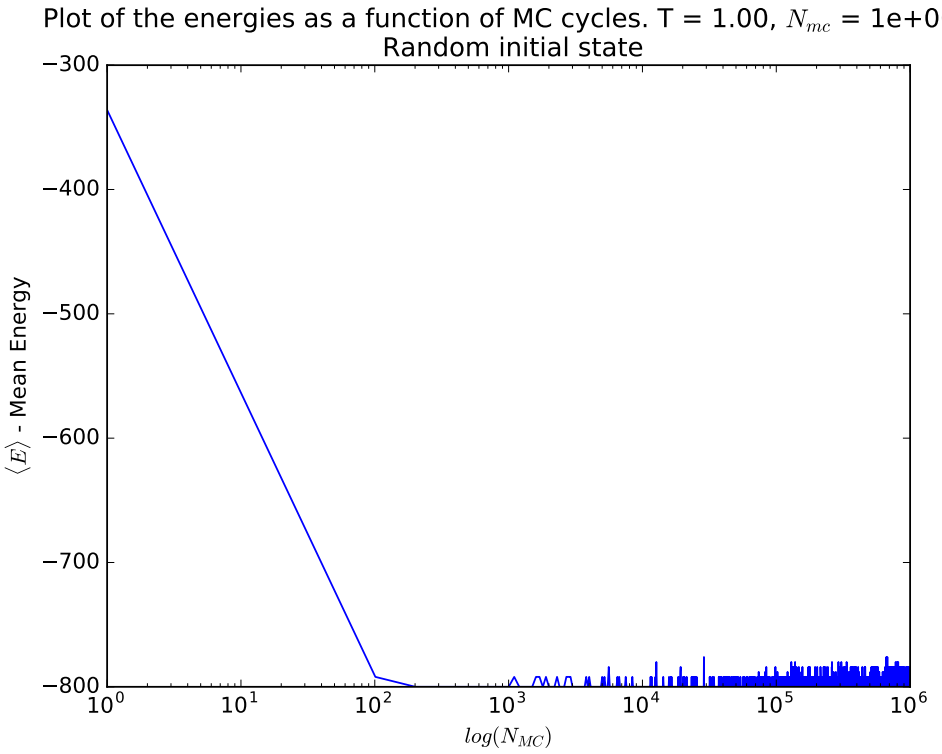

Figure 2: Stability of the energy, with $T = 1.0$ as a function of time. Logarithmic x-axis. Random initial state.

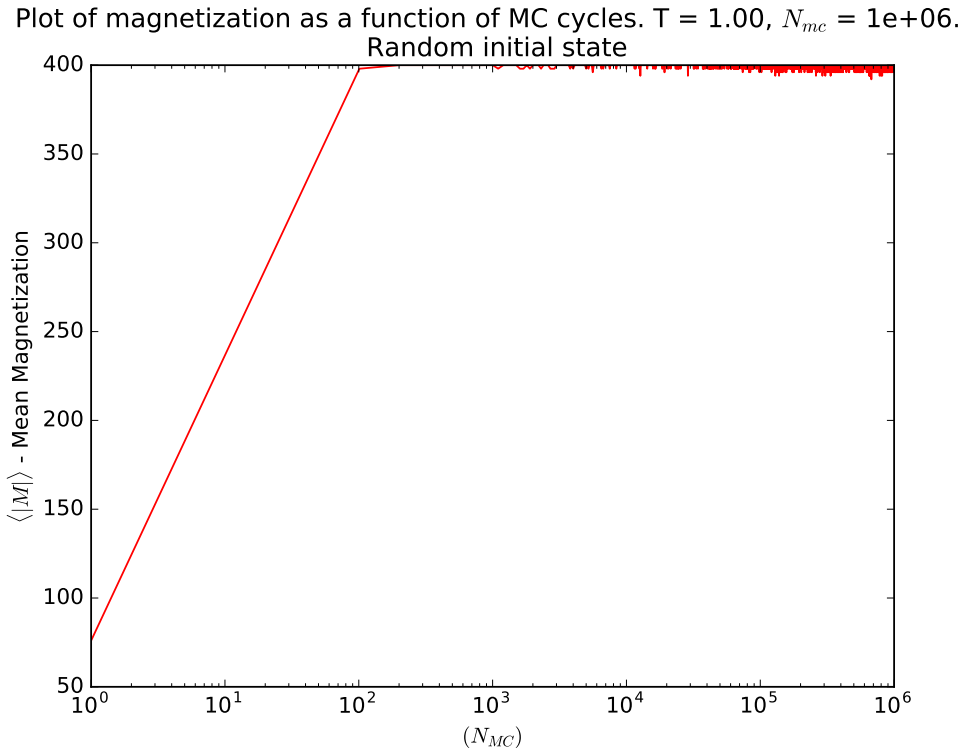

Figure 3: Stability of the magnetization, with $T = 1.0$ as a function of time. Logarithmic x-axis. Random initial state.

We now increase the temperature to $T = 2.4$. Figure 4 and 5 again shows how the energy and magnetization stabilizes respectively. This time, however, there are a lot more fluctuations for both parameters. This is not surprising. When we have a higher temperature, the system can reach a higher energetic state. Because of this, the spins will have the ability to change more (that is, more frequent flips) compared to the lower temperature case. Also note that the energy oscillates around $\langle E \rangle = -500$ instead of $\langle E \rangle = -500$ (for the lower energy case). Again, due to an increase in temperature, the system will therefore be in an excited state, which means it has a larger energy on average.

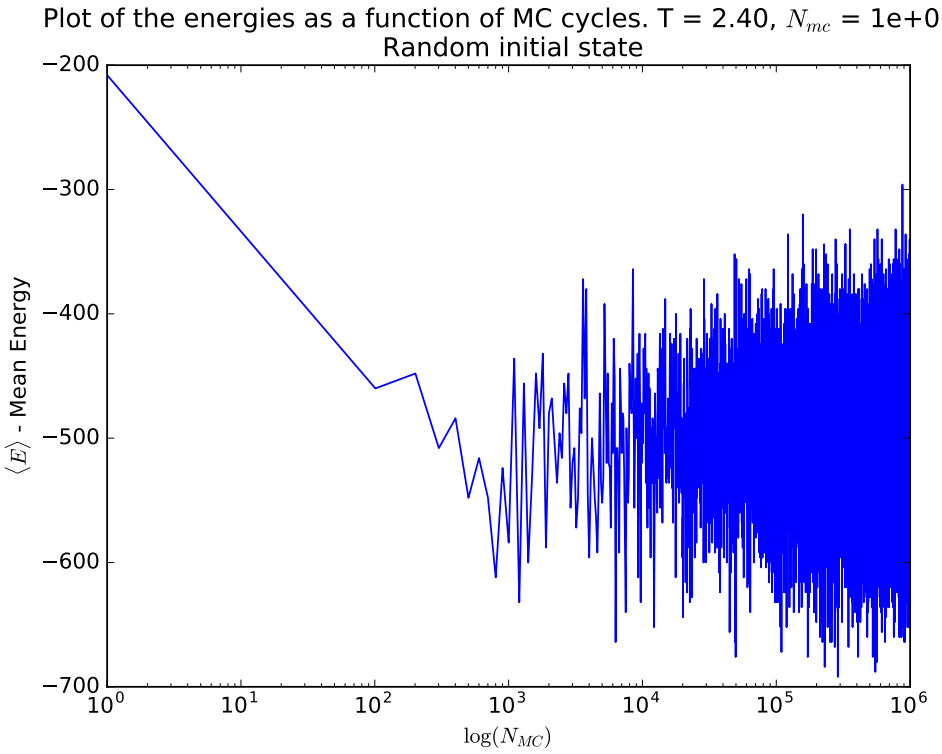

Figure 4: Stability of the energy, with $T = 2.4$ as a function of time. Logarithmic x-axis. Random initial state.

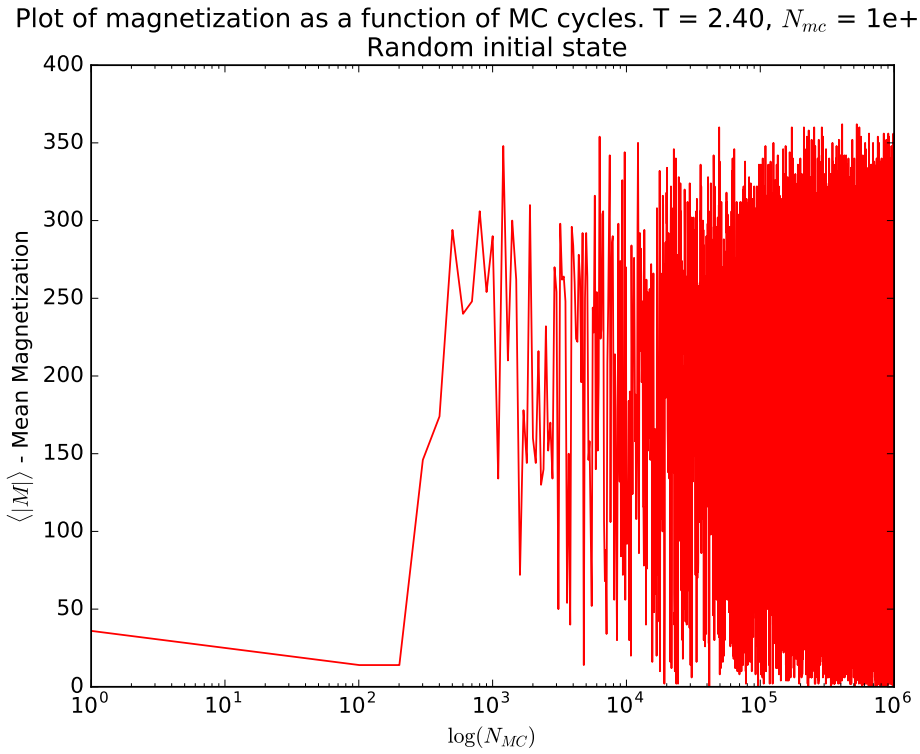

Figure 5: Stability of the magnetization, with $T = 2.4$ as a function of time. Logarithmic x-axis. Random initial state.

We can also have the program count how many times the a new configuration have been accepted. In figure 6, we see that we accept a lot more configurations as the temperature increases. The y-axis for this plot is in logarithmic scale. Note that I have only plotted two temperature points in the plot, so the number of accepted configurations may or may not increase linearly with temperature. This result agrees with the stability plots for $T = 2.4$ that we saw earlier.

Figure 7 shows how the number of accepted configurations increases over time. The y-axis is once again logarithmic. Once again, we see that the number of accepted configurations is higher for a higher temperature. Notice how we accept a lot more configuration changes at early times, and as time passes, we accept less. This indicates that the system is going towards

equilibrium.

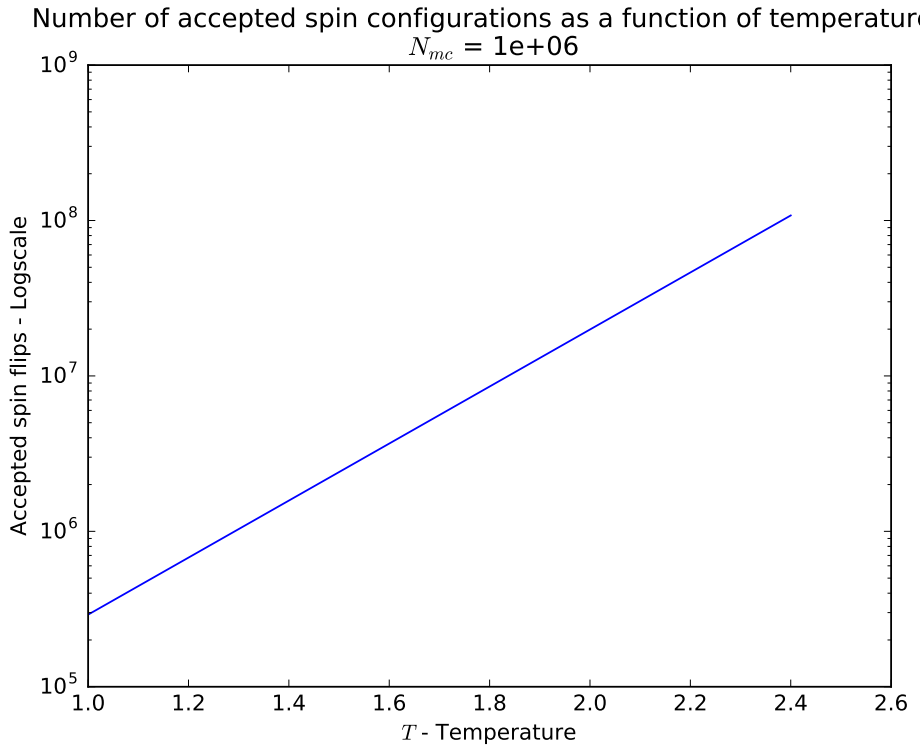

Figure 6: Number of accepted configurations for the two different temperatures. Logarithmic y-axis. Shows only two temperature points, so the increase may not be linear.

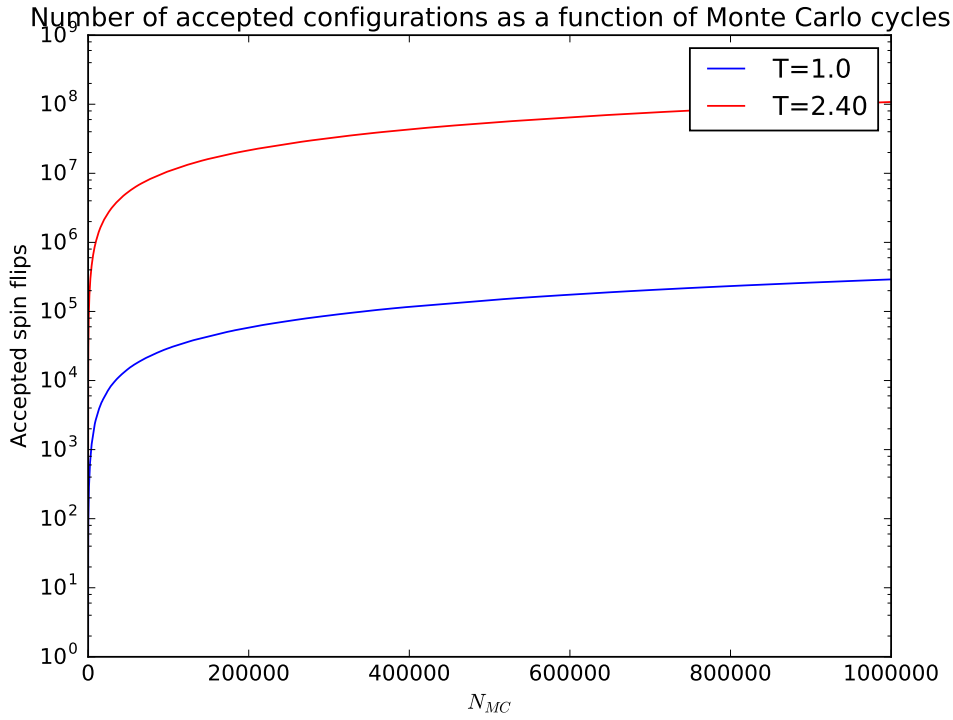

Figure 7: Number of accepted configuration changes over time. Logarithmic y-axis. We see that the number of accepted configurations increases as the temperature increases

If we now let all spins in the initial state be up spins, we would expect that the system will reach equilibrium a lot faster than a random initial state. The reason is that the energy will be at the lowest for a system with only up spins (also for down spins), which was shown previously in table 2. Because of this, the energy (as well as the magnetization) will already start near their equilibrium value.

This can easily be seen in figure 8 and 9 for the energy and magnetization respectively. Let us look at the plot for energy as an example. The initial energy is very close to the lowest possible energy for our system. On the other hand, in figure 2, the initial energy was larger due to an random initial state. The system will therefore reach an equilibrium state much faster when

the system starts will all spins up.

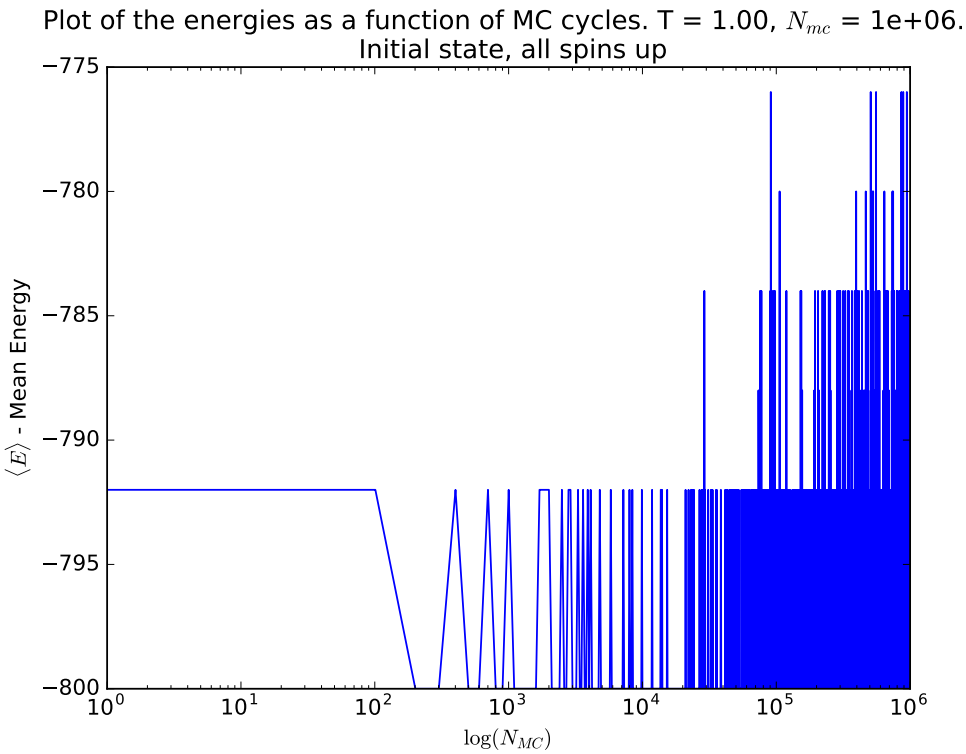

Figure 8: Stability for the energy for an initial state where all spins points up, with $T = 1.0$. Notice the lower initial energy, which is already close to the equilibrium energy.

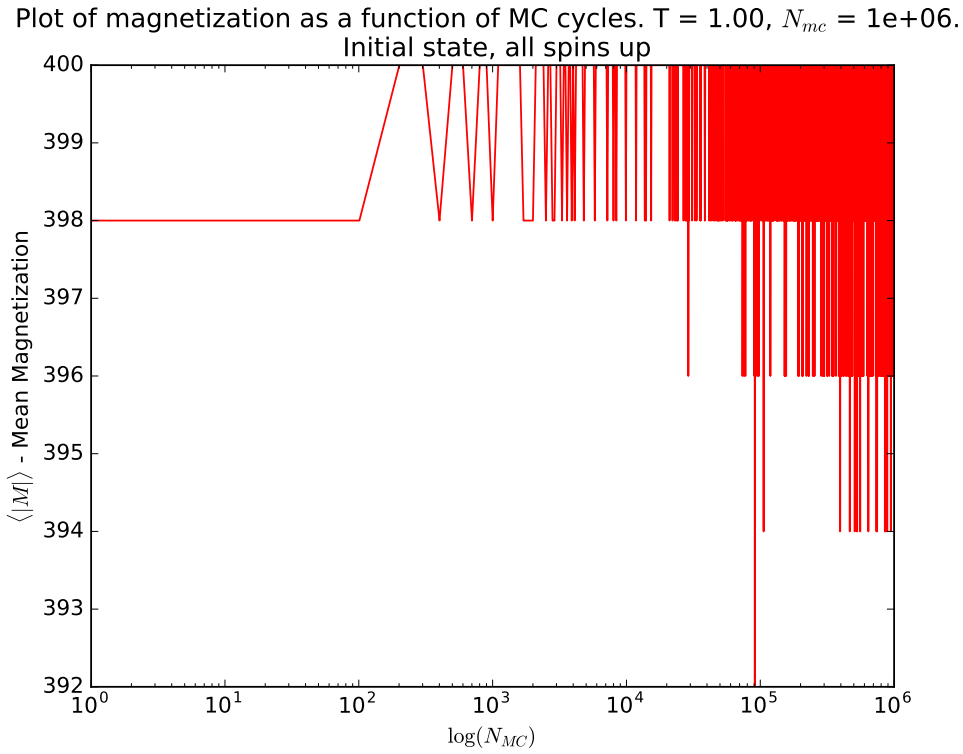

Figure 9: Stability of the magnetization for an initiale state where all spins points up, with $T = 1.0$. Like the energy case, the magnetization has an initial value close to equilibrium.

We can also look at the case with $T = 2.4$. Again, since the temperature is higher, the system can reach many more states compared to the lower energy case. Recall that that the spin state will change it's configuration much more frequently for larger temperatures, so we can also expect more oscillations for the energy and magnetization in this system, where all spins starts as up spins.

In figure 10, we can see that the energy oscillates around $\langle E \rangle = -500$. This is also not surprising due to a larger temperature, which means that the system will be in an excited state. Similar result can be seen for the magnetization in figure 11.

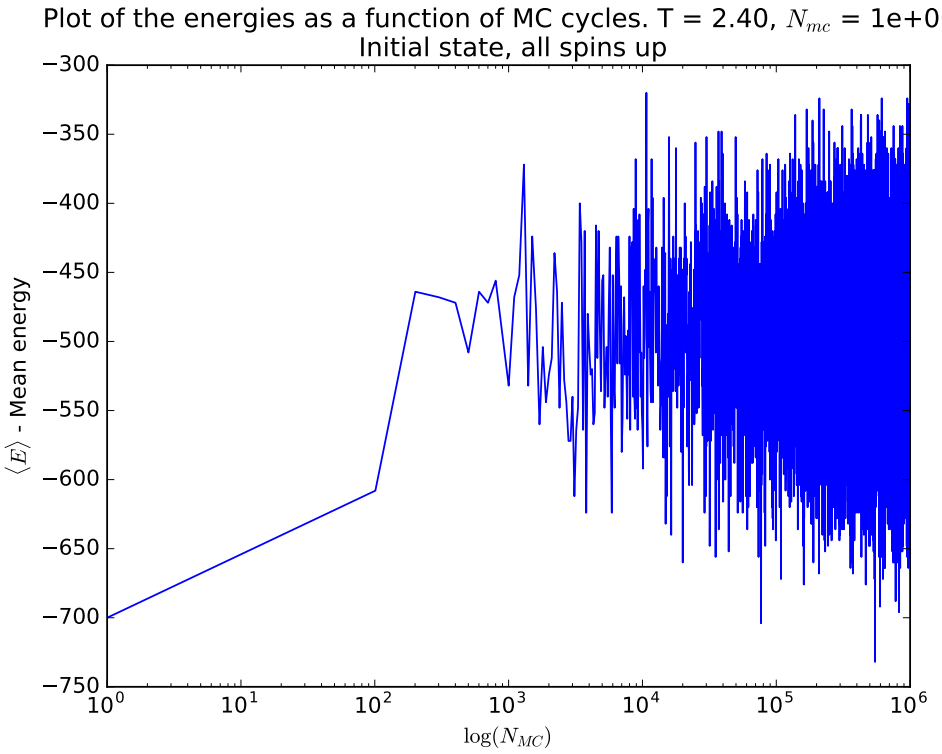

Figure 10: Stability for the energy for an initial state where all spins points up, with $T = 2.4$.

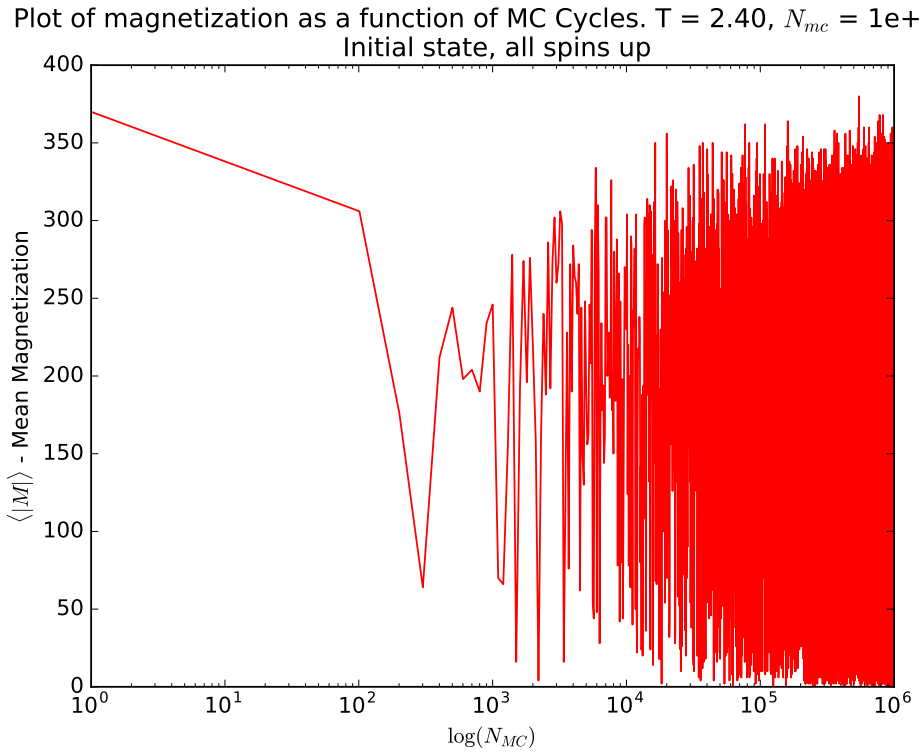

Figure 11: Stability of the magnetization for an initiale state where all spins points up, with $T = 2.4$.

4.3 Probability distribution

Instead of plotting how the energy evolves over time (as we did in the previous part), we can instead plot the probability distribution of the energy for different temperatures. We know that there is a higher probability of the energy being larger when the temperature is larger. Let us try to count how many times an energy has occurred during the computation and then plot a histogram of the result.

A plot of the probability distribution for both $T = 1.0$ and $T = 2.4$ can be found in 12. As we can see, when the temperature is lower, the energy has a

higher probability of being in a lower energetic state. When the temperature is larger, the energy can be at a more energetic state. Plots where the histograms are separated can be found in the appendix.

Although, the standard deviation of the histogram, for $T = 2.4$, larger than the case of $T = 1.0$. What this means is that the system for $T = 2.4$ has many more energy states to choose from, and each of them has roughly the same probability. As we mentioned in the previous part, the system can reach many more states when it has a higher temperature, which corresponds to different energies. The result of this also agrees with the results from figure 4 and 10. The energies for those cases were oscillating around $\langle E \rangle = -500$, and the histogram shows the exact same result.

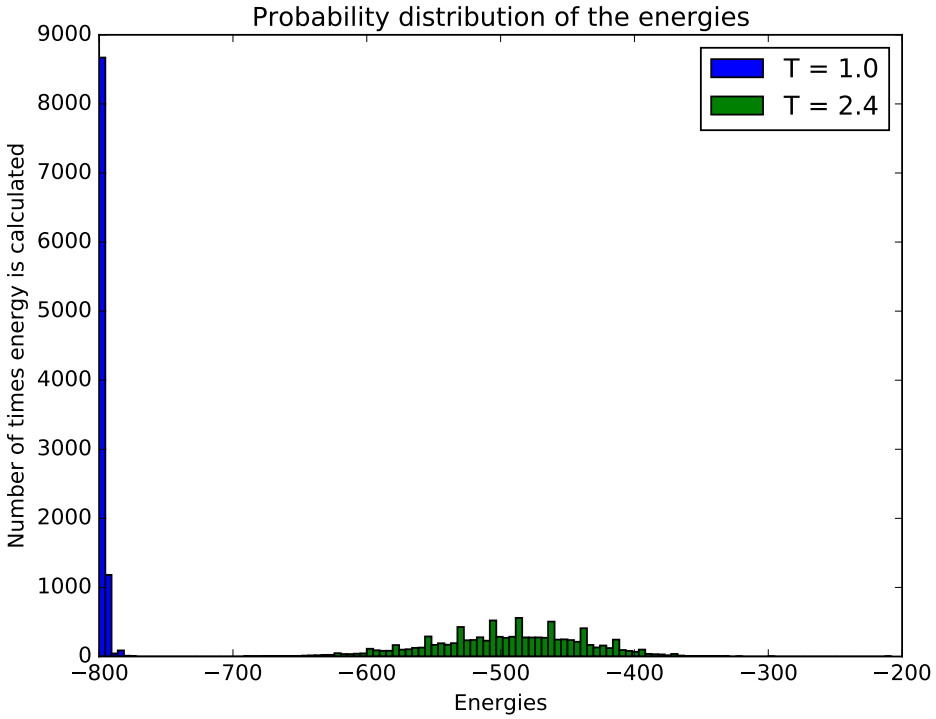

Figure 12: Probability distribution of the energies for $T = 1.0$ (blue) and $T = 2.4$ (green).

We are also tasked to compute the variance of the energies. I have computed the standard variance from the C++ program and I will compare it to the variance given from the Python (Numpy) function *numpy.var*. This is done by saving all the energies (that was counted in the C++ program) in an array and then just call the variance function from Numpy. The results are found in the following output:

```
Computed variance =  10.2423 , for T = 1.0
Computed variance =  3214.74 , for T = 2.4
Numpy variance =  31.11786096 ,for T = 1.0
Numpy variance =  3121.04932864 ,for T = 2.4
```

The variance for $T = 2.4$ from Numpy is very close to the computed value. However, they are not quite in agreement for $T = 1.0$. The reasons are unknown to me, and I really doubt that there are any errors in the Numpy function, which means that there has probably been a computation error from my part. As we saw from the histogram, the system with a higher temperature had more energies to choose from. This results to a larger variance, which is exactly what got.

4.4 Phase transitions

5 Conclusion

As we saw from the results, the Metropolis algorithm is a solid Monte Carlo algorithm. The computed value, when we compared it to a simple 2×2 lattice, were incredibly close to the analytical one.

The algorithm also worked very well when we increased the lattice size. We saw that the computed energy were indeed larger for a larger temperature. This agrees with our physical intuition, where a system with a higher temperature is more energetic than a system with lower temperature.

However, the results of the phase transitions and critical temperature may or may not be completely consistent. The system were expected to be very

volatile near the critical temperature for a larger lattice, but we used very few Monte Carlo samplings near this region. The results may therefore not be 100% correct.

Although, one can probably bump the number Monte Carlo cycles for $L = 140$ and even reduce the temperature step to obtain a more consistent answer, but that will increase the computation time quite considerably.

6 Appendix

All the figures given in the appendix can also be found in the GitHub page.

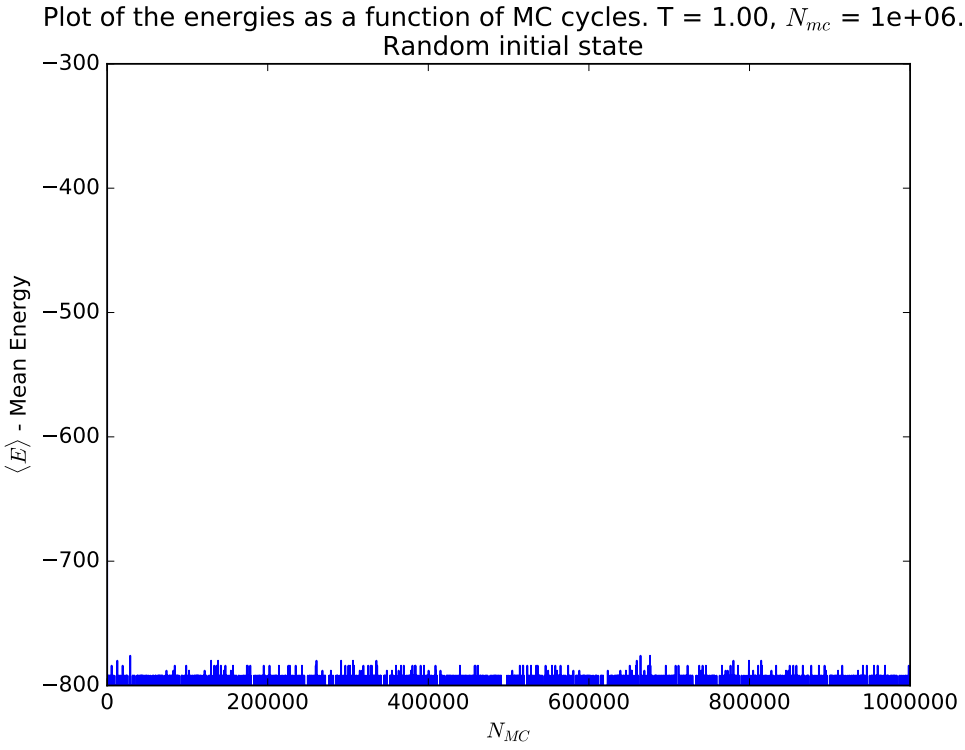

Figure 13: Stability of the energy for T = 1.0.

References

[1] M. Hjorth-Jensen, *Computational Physics*, 2015, 551 pages