

# Simulation of the 2D Ising Model

Mark Van Selous

UID: 116031813

December 15, 2020

## Part I: Random Spins and Magnetization

### Introduction

The Ising model was created to study a phase transition from paramagnetism to ferromagnetism. It does so by simulating the magnetic dipole moments of atomic spins.

In this report, we will outline how we recreated and applied the 2D Ising Model with python. We first used the Ising Model to calculate the magnetization of randomly generated matrices and the dispersion of a large sample of them. Then, we used the Metropolis algorithm to solve for the average energy and magnetization (after thermalization). More specifically, we experimented with a variable  $\beta = 1/kT$  to learn how the average energy and magnetization evolve as a function of temperature.

The body of our paper includes code snippets to help the reader follow along with our work. Please see the appendixes for the highlights of the project. You can also access our python script here: [https://colab.research.google.com/drive/1q3xJbQ1UqtFf9qWAC-XtnDKa9E1\\_DAyJ?usp=sharing](https://colab.research.google.com/drive/1q3xJbQ1UqtFf9qWAC-XtnDKa9E1_DAyJ?usp=sharing). We have also attached its pdf printout with this report.

### Generating the matrices

To generate a  $N = m \times n$  matrix of spin values, we wrote a function that generates a random  $m$  by  $n$  matrix of 0s and 1s and remapped the 0s to -1s. We could then find that system's magnetization,  $m = \frac{1}{N} \sum_{i=1} \sum_{j=1} \sigma_{i,j}$ , by summing the spin of each element and dividing by the total number of elements. We could estimate the dispersion,  $\overline{\Delta m^2} = \overline{m^2} - \overline{m}^2$ , by generating the large sample of matrices and making use of our calculation for each systems magnetization.

```

1 def Generate_Spins(N):
2     matrix = np.random.randint(2,size=(N))
3     matrix[matrix == 0] = -1
4     magnetization = np.sum(matrix)/np.prod(N)
5     return {"matrix":matrix, "mag":magnetization}

```

Figure 1: The function, `Generate_Spins(N)`, we wrote to generate an  $N = m \times n$  matrix and compute its magnetization. See Appendix A for sample matrices we generated.

Especially with the smaller sized matrices, we found there to be a high degree of variability in the dispersion among the samples we generated. This was due to the fact that we were finding the dispersion of a much large sample of spins in the larger matrices. Since every spin had an equal chance of being spin up or spin down, we would expect for the distribution of magnetization's to be more starkly concentrated around  $m = 0$ .

As the number of spins increases, we would expect the average magnetization,  $\overline{m}$  to tend towards zero. While the term  $\overline{m^2}$  will also tend towards zero, it would do such much more slowly as every matrix makes a non-negative contribution to it.

```

1 def Calc_Dispersion(N,steps):
2     m_squared_bar = m_bar = 0
3
4     for idx in range(0,steps):
5         m = Generate_Spins(N)["mag"]
6         m_bar += m; m_squared_bar += m**2
7
8     for item in (m_bar,m_squared_bar): item = item/steps
9
10    return {"dispersion": m_squared_bar - m_bar**2,
11            "m_squared_bar":m_squared_bar,
12            "m_bar":m_bar,}

```

Figure 2: The function, `Calc_Dispersion(N,steps)`, we wrote to estimate the dispersion with a sample of  $n = \text{steps}$  matrices. Please see Appendix B for the results of running this function for samples with three different sized matrices.

## Part II: Metropolis Algorithm and the Ising Model

### Implementing the Metropolis Algorithm

Once we developed our method for generating spin matrices, we created our implementation of the Metropolis-Hasting algorithm. We will briefly describe

the iterative steps of this model.

First, we generated a random spin matrix. Like the spin matrices we previously generated, its elements are completely random and do not reflect a thermalized system. Second, we randomly select a spin element and calculate the energy difference,  $\Delta E_{2D} = 2J\sigma_{i,j}(\sigma_{i-1,j} + \sigma_{i+1,j} + \sigma_{i,j-1} + \sigma_{i,j+1})$ , of flipping that spin. Third, if the resulting energy difference is less than 0, we flip the spin. If the energy difference is positive, but a random  $X$  (taken from the standard uniform distribution) is less than  $e^{-\beta\Delta E_{2D}}$  we still elect to flip the spin. In practice, we used the inequality  $\ln(x) < -\beta\Delta E_{2D}$ . Forth, we calculate the system's properties we are interested in and factor them into a running average. We chose to calculate the systems energy  $E(r) = -J \sum_{ij} \sigma_i \sigma_j$  and magnetism. These four steps were repeated a sufficient number of times so that the average energy would converge to oscillate closely around a specific value.

```

1 def Metropolis_Sampling(N,steps,beta):
2     x_max,y_max = N; idx = M_idx = E_idx = 0; bar_M = []; bar_E = []
3
4     #1 Generate Spins
5     spins = Generate_Spins(N)["matrix"]
6     while idx < steps:
7         idx += 1
8         #2a Pick a spin at random
9         x,y = np.random.randint(0,x_max),np.random.randint(0,y_max)
10        #2b Calculate Energy difference
11        dE = Energy_Difference(spins,x,y,x_max,y_max)
12        #3a If dE < 0, flip that spin
13        if dE < 0:
14            spins[x,y] *= -1
15        #3b Else, if a random X < e^(-beta dE) flip spin
16        elif (np.log(np.random.uniform(0,1)) < -beta*dE):
17            spins[x,y] *= -1
18        #3c if not, keep in unchanged
19        #4a Calculate barE
20        E = Compute_Energy(spins,x_max,y_max)
21        E_idx += E; bar_E.append(E_idx/idx)
22        #4b Calculate barM
23        M_idx += np.sum(spins)/np.prod(N); bar_M.append(M_idx/idx)
24        #5 Providing an update throughout the cycle
25        if idx % (steps*0.1) == 0:
26            print("***** \n Iterations:",
27                  idx, " ("+"{:.0%}".format(idx/steps), "complete)",
28                  "\n Average Energy:", bar_E[idx-1],
29                  "\n Average Magnetization:",bar_M[idx-1],
30                  "\n*****")
31
32    return {"bar_E":bar_E,"bar_M":bar_M,"beta":beta}

```

Figure 3: The function, Metropolis\_Sampling(N,steps,beta), we wrote to perform the Metropolis-Hasting algorithm. Here  $\beta = 1/kT$  is used in calculating the Boltzmann factor and can be varied to simulate systems at different temperatures.

```

1 def Energy_Difference(M,x,y,x_max,y_max):
2     dE = 2*M[x,y]*(M[(x-1)%x_max,y]+M[(x+1)%x_max,y]+
3         M[x,(y-1)%y_max]+M[x,(y+1)%y_max])
4     return dE
5
6 def Compute_Energy(M,x_max,y_max):
7     E = 0
8     for x in range(0,x_max):
9         for y in range(0,y_max):
10             E -= M[x,y]*(M[(x+1)%x_max,y]+M[x,(y+1)%y_max])
11     return E

```

Figure 4: The functions, `Energy_Difference(M,x,y,x_max,y_max)` and `Compute_Energy(M,x,x_max,y_max)` which are called as part of the Metropolis Sampling algorithm. The borders of the input matrix are nearest neighbors under modular arithmetic.

## Results

We initially applied our algorithm to  $3 \times 3$ ,  $10 \times 10$ , and  $64 \times 64$  spin matrices representing systems with  $\beta = 0.1, 0.2$ , or  $0.4$ . With a fixed  $\beta$ , we found the average energy to be directly proportional to the number of spin elements. For example, a  $10 \times 10$  matrix has a little more than 11 times as many elements as a  $3 \times 3$  matrix. We similarly found the average energy to be roughly by the same proportions. This result was not too surprising and served as a good indicator for our models reliably.

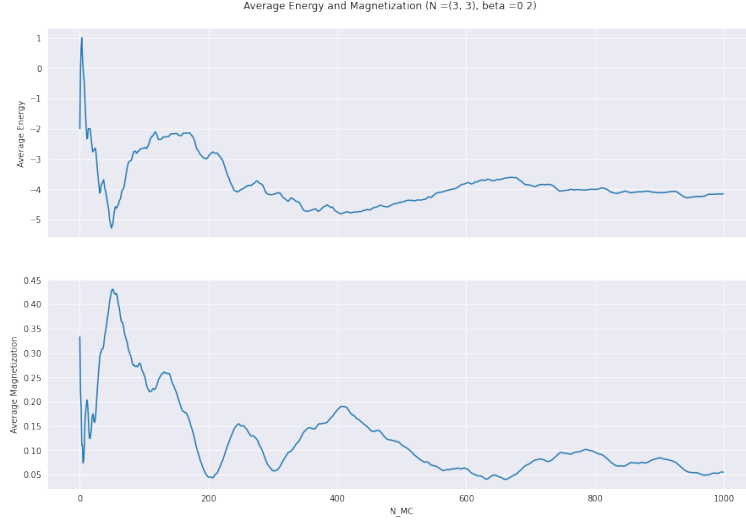


Figure 5: An example plot of both the average energy and magnetization vs  $N_{MC}$ . Please see Appendix C for our selected examples or the python script to explore some more.

### Temperature Effects on Average Energy

It also allowed us to further investigate the the average energy while sticking with systems with a common dimension. Our next step was to compute the average energy with an assortment of values for  $\beta$ . To help smooth out the results, we averaged the results from five trials which themselves were averages of the last 100 values obtained by the model.

When we did so, we found the average energy to approximately follow the lower half of an inverted sigmoid function as a function of  $\beta$ . But if we were to extend the model to the, non-physical, values of  $\beta < 0$ , the average energy remains approximately zero.

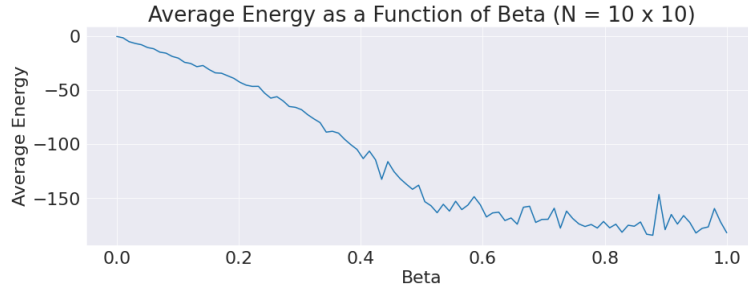


Figure 6: Plot of the average energy as a function of beta. There was a noticeably larger amount of variance in our data as beta increased.

### Temperature Effects on Average Magnetization

We followed a similar procedure to learn how the average Magnetization evolves as a function of  $\beta$ . But if we were to average the results from multiple trials, we would expect to obtain zero. So instead, we made a scatter plot depicting the results from all of the individual trials.

We found that around  $\beta = 0.3$  the average magnetization ballooned out in either direction. Then, by  $\beta = 0.6$ , reached its maximum and minimum of  $-1$  and  $1$ . At these high levels for  $\beta$  there were few trials that managed to meaningfully move towards zero by the end of 5000 interactions of the Metropolis Algorithm.

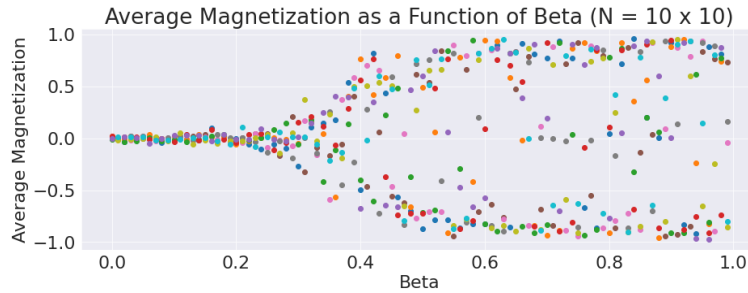


Figure 7: Plot of the average magnetization as a function of beta. Each trial was performed with 5000 iterations so that we could compare how quickly the average magnetization would converge to zero. Seeing the greater spread in the average magnetization helps to explain the variance in the average energy as beta increases.

### Conclusion

We were successful in using the Metropolis algorithm to simulate thermalization in the 2D Ising model. We found the average energy in the system to be directly

proportional to the number of spin elements and graphed a relationship between the average temperature and  $\beta$ .

It was more challenging to draw any direct conclusions about the average magnetization. We did find that for a fixed number of iterations of our algorithm, the average magnetization balloons away from zero as beta increases (particularly for  $\beta > 0.3$ ). This had a noticeable impact on the average energy by increasing the variance in our results.

## Appendix A: Generated Spin Matrices

```
[3] 1 three_ex = Generate_Spins((3,3))
    2 print("Matrix: \n", three_ex["matrix"], "\n \n Magntization: \n", three_ex["mag"])

Matrix:
[[-1 -1 1]
 [-1 -1 1]
 [ 1 1 -1]]

Magntization:
-0.11111111111111111

[4] 1 ten_ex = Generate_Spins((10,10))
    2 print("Matrix: \n", ten_ex["matrix"], "\n \n Magntization: \n", ten_ex["mag"])

Matrix:
[[-1 1 -1 -1 -1 -1 1 -1 1 1]
 [ 1 1 1 1 -1 -1 1 -1 1 1]
 [-1 -1 1 -1 -1 -1 1 1 1 1]
 [-1 -1 -1 1 1 -1 1 -1 -1 1]
 [-1 -1 1 -1 -1 1 1 1 -1 -1]
 [ 1 -1 1 -1 1 1 1 -1 1 1]
 [ 1 -1 1 -1 1 -1 -1 1 1 -1]
 [-1 -1 1 -1 -1 1 -1 1 1 1]
 [ 1 1 1 -1 -1 1 1 1 1 1]
 [ 1 -1 -1 -1 -1 1 -1 -1 1 1]]

Magntization:
0.06

[5] 1 big_ex = Generate_Spins((64,64))
    2 print("Matrix: \n", big_ex["matrix"], "\n \n Magntization: \n", big_ex["mag"])

Matrix:
[[ 1 1 -1 ... -1 1 -1]
 [-1 -1 -1 ... -1 -1 -1]
 [ 1 1 -1 ... 1 1 -1]
 ...
 [-1 -1 1 ... -1 -1 1]
 [-1 1 -1 ... 1 -1 -1]
 [ 1 1 -1 ... -1 1 -1]]

Magntization:
-0.00146484375
```

Figure 8: An example  $3 \times 3$ ,  $10 \times 10$ , and  $64 \times 64$  spin matrix generated, along with its magnetization, by our `Generate_Spins(N)` function.

## Appendix B: Dispersion Calculations

```
▼ Dispersion of a  $N = 3 \times 3$  matrix

[7] 1 N, steps = (3,3), 100000
    2 print(sorted(Calc_Dispersion(N,steps).items()))

[('dispersion', 11096.543209876876), ('m_bar', 2.0000000000003793), ('m_squared_bar', 11100.543209876878)]

[8] 1 N, steps = (3,3), 100000
    2 print(sorted(Calc_Dispersion(N,steps).items()))

[('dispersion', 7322.469135803185), ('m_bar', 61.333333333332604), ('m_squared_bar', 11084.246913580873)]

▼ Dispersion of a  $N = 10 \times 10$  matrix

[9] 1 N, steps = (10,10), 100000
    2 print(sorted(Calc_Dispersion(N,steps).items()))

[('dispersion', 840.6328000006647), ('m_bar', -12.679999999999756), ('m_squared_bar', 1001.4152000006585)]

[10] 1 N, steps = (10,10), 100000
    2 print(sorted(Calc_Dispersion(N,steps).items()))

[('dispersion', 771.9440000006405), ('m_bar', -15.139999999999933), ('m_squared_bar', 1001.1636000006384)]

▼ Dispersion of a  $N = 64 \times 64$  matrix

[11] 1 N, steps = (64,64), 100000
    2 print(sorted(Calc_Dispersion(N,steps).items()))

[('dispersion', 24.33634901046753), ('m_bar', 0.4130859375), ('m_squared_bar', 24.506989002227783)]

[12] 1 N, steps = (64,64), 100000
    2 print(sorted(Calc_Dispersion(N,steps).items()))

[('dispersion', -24.454711437225342), ('m_bar', 6.99365234375), ('m_squared_bar', 24.456461668014526)]
```

Figure 9: Example dispersion calculations with our three sized matrices. From these calculations, we were able to infer that the dispersion would decrease as the dimensions of  $N$  increase. We also outputted  $\overline{m}$  and  $\overline{m^2}$  and noticed much greater variance in  $\overline{m}$ .

## Appendix C: Results of the Metropolis Algorithm

The following plots are compliments to figure 5 in the text above.

Th first two plots were generated with a constant  $\beta = 0.2$  but variably sized matrix. They illustrate how the average energy is directly proportional to the number of spin elements.

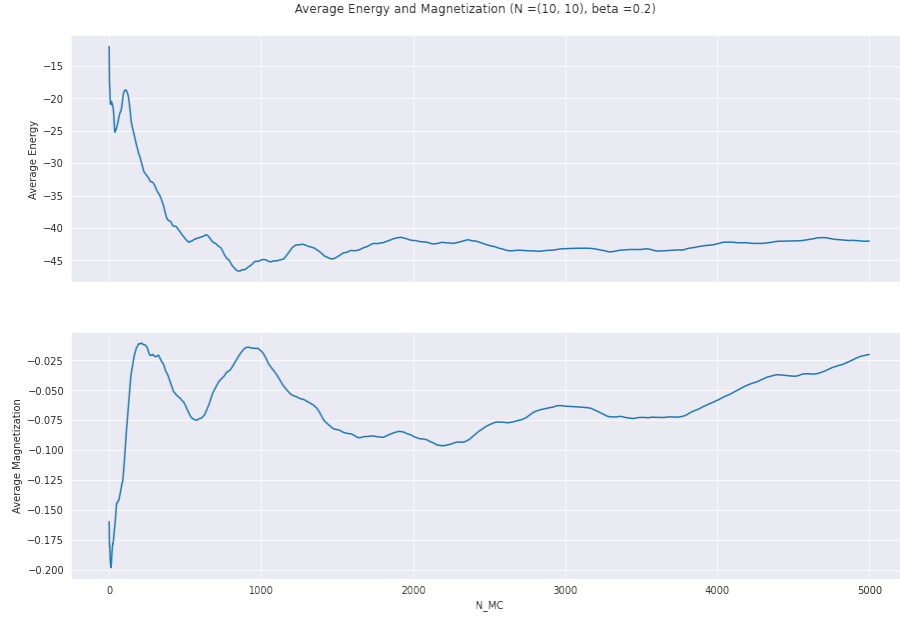


Figure 10: The final values for the average energy and magnetization were -42.0168 and -0.020 respectively. Although the average energy did not significantly change after  $N_MC = 2000$ , the average magnetization started to drift towards 0.



Figure 11: In this trial, the last recorded values for the average energy and magnetization were -1643.559 and -0.012 respectively. This time the average magnetization gradually decreased throughout most of the process.

The last two plots were generated with a constant  $N = 3 \times 3$  but differing  $\beta$ .

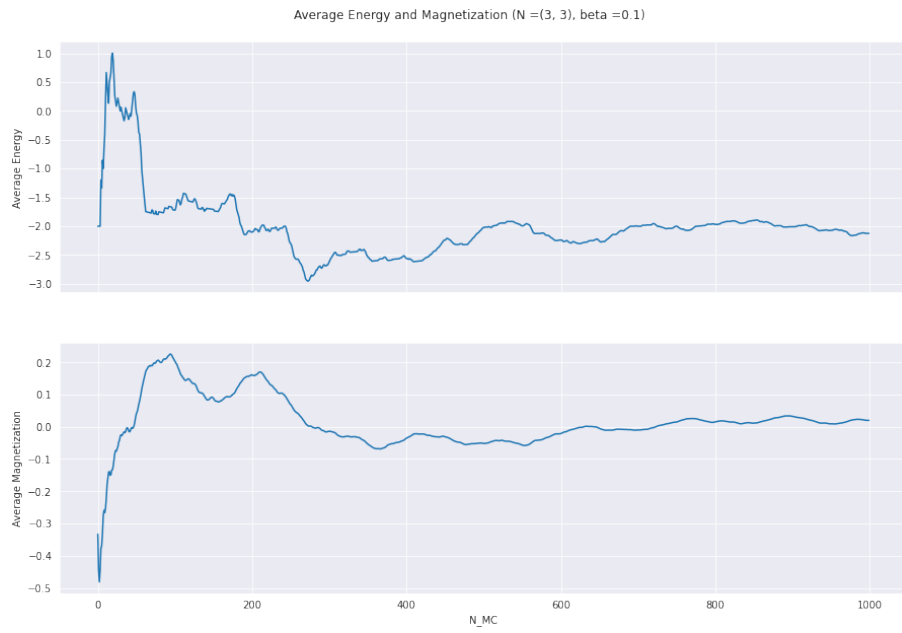


Figure 12: In this trial (with  $\beta = 0.1$  and  $N = 3 \times 3$ ) the final values for the average energy and magnetization were -2.124 and 0.0200.

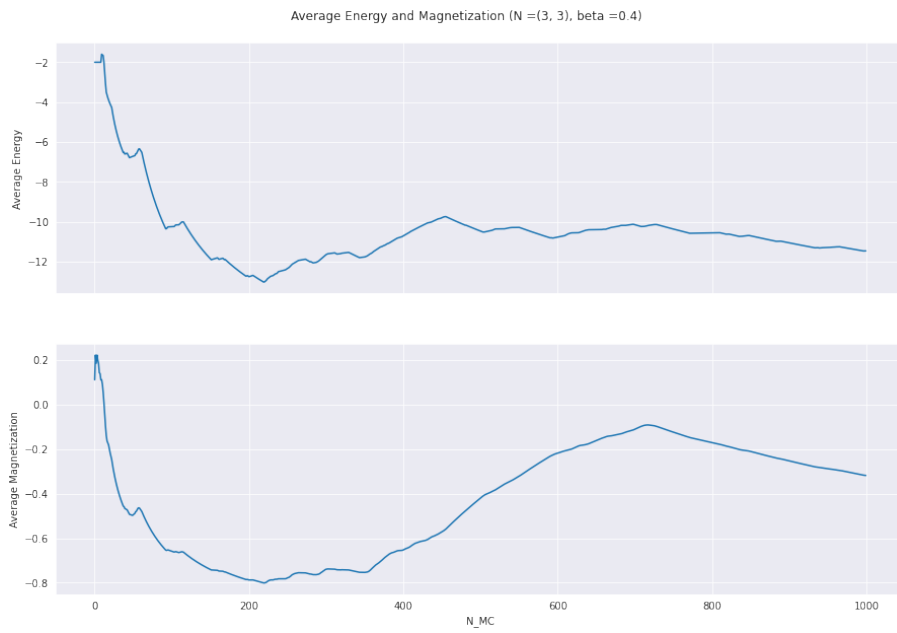


Figure 13: In this final example, the average energy and magnetization ended up at -11.452 and -0.318. This magnetization was noticeably higher than the trials with a lower  $\beta$ .