Monte-Carlo Simulations of the 2D Ising Model

Sumer Malhotra, Olivia Jackson, Saad Mohiuddin, Patrick Li

Machine Learning in Physics, PHYS3151

Abstract - A Monte Carlo simulation of the Ising model on a 2D square lattice with no external magnetic field was performed for various lattice sizes at different temperatures. The energy and magnetization were calculated and plotted against temperature for the largest lattice size, d=32. The critical temperature was calculated to be 2.28 K. The influence on the phase transition of the Gaussian noise in the spin coupling values was demonstrated.

I. Introduction

The Ising model is a lattice model modeling magnetism in materials using the configurations of magnetic dipole moments of atoms. A configuration tends towards the lowest energy state. Due to thermal fluctuation, there is a slight chance of the state changing to a higher energy state with probability proportional to:

$$e^{\frac{-\delta E}{T}} \qquad \textbf{(1)}$$

The order parameter M, which also denotes the mean magnetization, is positive for low temperatures, and zero for high temperatures. At a certain temperature, the shape of the M/T graph changes drastically. This is known as the critical temperature.

II. Experimental Method

In the model, energy change was considered to be entirely due to the interactions of atoms with neighbor atoms. A random lattice consisting of +1 and -1 magnetizations was generated. An atom was chosen and the energy change for a flip was considered. If the change was negative, the flip was given a probability of 1 and if positive it was given the probability from Eqn. 1. The algorithm was run for 10^7 iterations at which time the lattice had converged to an approximately stable configuration with Gaussian noise considered. The second half of the data set was taken and averaged to find energy and magnetization values for each temperature. Graphs for energy and magnetization were then plotted and data was collected for other lattice sizes.

III. RESULTS

First graphs were produced for a 16 x 16 and 32 x 32 lattice showing the energy and magnetization for 60 different temperature values. These can be seen in Figure 3 and 4. Both graphs followed the expected shape. The program was then run for all lattice sizes for 10^7 iterations (Figures 3 & 4 bottom graphs). The critical temperature was calculated to be 2.28 by taking the average over values of T where E witnessed a large drop.

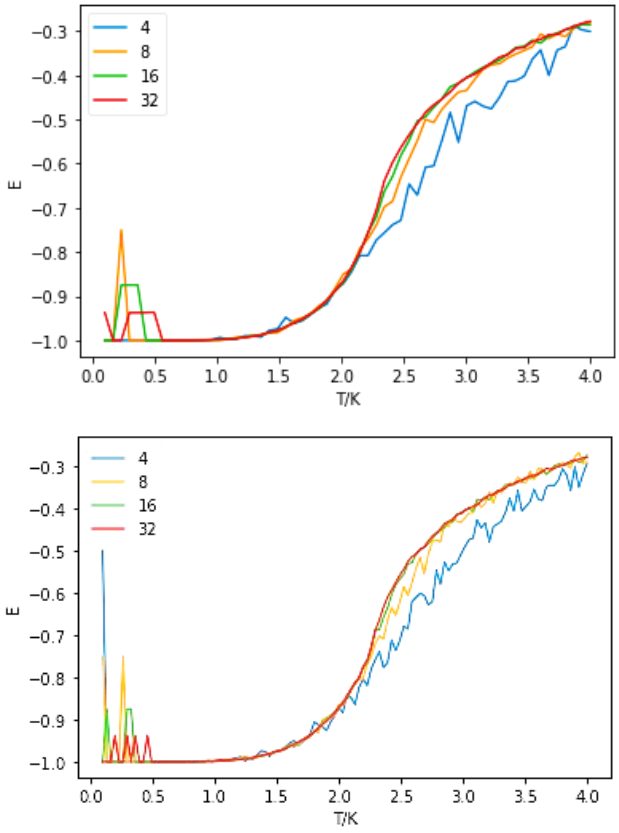


FIGURE 1 The graph of energy against temperature for lattice sizes of 4, 8, 16 and 32. The shape of the graph is consistent with theoretical predictions and is smoother for larger lattice dimensions. The graph on the bottom was the result of a larger number of iterations of T.

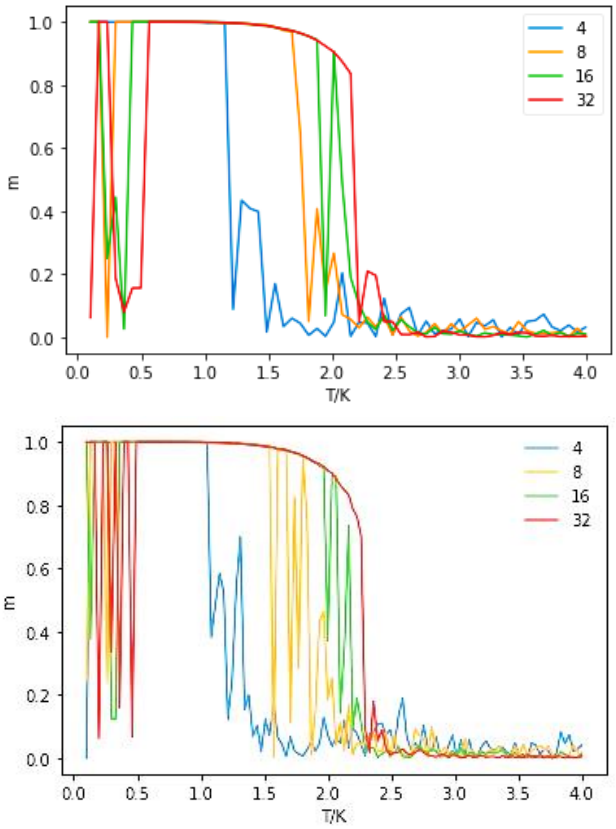


FIGURE 2 The graph of magnetization against temperature for lattice sizes of 4, 8, 16 and 32. The shape of the graph is consistent with theoretical predictions however there is a lot of noise for smaller lattices. The graph on the bottom was the result of a larger number of iterations of T.

IV. Discussion

Figure 2 shows a clear change in shape, which denotes the shape transition. Initially the algorithm was designed to calculate the energy of the lattice after each proposed change of state and the probability of accepting the change after each iteration. This meant doing an exponential operation for each iteration which resulted in the program being computationally expensive and taking an unacceptable amount of run time. To overcome this, since changing a state only affected the neighbor atoms, the change in energy could be calculated directly instead of the energy of the entire lattice. Additionally, as there were only 2 possibilities for positive ΔE (+4 and +8) this could replace the exponent function with a list of 2 values for each T. These changes significantly reduced run time of the program.

In the graphs, we see spikes in E and m near low values of temperature which can be attributed to sublattices of the system being stuck in highly ordered arrangements. At higher temperatures, this phenomenon does not take place.

V. Conclusion

Although a larger lattice size would result in a more accurate model, calculations were performed for a lattice of size 32 due to limitations in computing power. It was found that the lattice stabilized after a number of iterations and that the magnetization tended to zero. The graph of the energy of the system followed the characteristic shape shown by lattices at different temperature values. The critical temperature was found to be 2.28 which is consistent with the results of other models [3].

Acknowledgements

We would like to express our utmost gratitude to our professor, Dr. Zi Yang Meng and to our tutor, Jiarui Zhao from the Department of Physics for their continued guidance through this project.

References

[1] Physics.bu.edu, 2018. [Online]. Available: http://physics.bu.edu/~py502/lectures5/mc.pdf. [Accessed: 24- Nov- 2019].

[2] "Ising model", En.wikipedia.org, 2019. [Online]. Available: https://en.wikipedia.org/wiki/Ising_model. [Accessed: 24- Nov- 2019].

[3] Pdfs.semanticscholar.org, 2014. [Online]. Available: https://pdfs.semanticscholar.org/002a/23b6bc3f85f80dc74d075473b0f04746edbb.pdf. [Accessed: 24- Nov- 2019].

[4] Phas.ubc.ca, 2005. [Online]. Available: https://www.phas.ubc.ca/~berciu/TEACHING/PHYS503/PROJECTS/05_dominic.pdf. [Accessed: 24- Nov- 2019].

2D Ising Model

November 24, 2019

1 Simulating a 2D Ising Model with Monte Carlo Method

1.0.1 Sumer Malhotra (3035347457) | Olivia Jackson (3035653399)

1.0.2 Saad Mohiuddin (3035492989) | Patrick Li (3035654795)

1.1 Importing Libraries

[34]:
```
import numpy as np
import matplotlib.pyplot as plt
from random import *
import timeit
#run time calculation feature added.
```

1.2 User-defined Functions

[35]:
```
def spinGenerator():
    x = random()
    if x > 0.5:
        return 1
    return -1

def computeNeighbors(lattice, x, y):
    vals = [-1, 1]
    sum = 0
    for i in vals:
        if (x+i) % len(lattice) == 0 and x+i != 0:
            x1 = 0
        else:
            x1 = x+i
        if (y+i) % len(lattice) == 0 and y+i != 0:
            y1 = 0
        else:
            y1 = y+i
        sum += lattice[x1][y] + lattice[x][y1]
    return sum
```

```
def generateLattice(dimensions):
    return np.array([[spinGenerator() for i in range(dimensions)] for j in␣
 ↪range(dimensions)])

def energy(lattice, n):
    energy=0
    for x in range(0, n):
        for y in range(0, n):
            energy += lattice[x,y] * nsum(x, y, lattice)
    return -energy/2

def magnetization(lattice, n):
    return np.sum(lattice)/(n**2)

def nsum(x, y, lattice):
    vals = [-1, 1]
    sum = 0
    for i in vals:
        if (x+i) % len(lattice) == 0 and (x+i) != 0:
            x1 = 0
        else:
            x1 = x+i
        if (y+i) % len(lattice) == 0 and (y+i) != 0:
            y1 = 0
        else:
            y1 = y+i
        sum += lattice[x1][y] + lattice[x][y1]
    return sum

average_E = [[],[],[],[]]
average_m = [[],[],[],[]]
rescale_E = [[],[],[],[]]
T_list = list(np.linspace(T_min, T_max, num=temp_iterations) )
```

1.3 Simulation of the lattice at a Temperature, T

[36]:
```
def run():

    samplerate = int(1E5)
    #this mean sample the m&E at the moment for every 'smaplerate' iteration
    operation = int(3E7)
    iteration = range(operation)
    #the total iteration for one temperature
    alpha = 0.5
    #the portion of the data at the end of each cycle with particular T for the␣
 ↪calculation of average m&E
```

```
    temp_iterations = 120
     #taking more datapoints near the critical temperature.
    #the number of temperature assess in the programme
    T_max = 4
    T_min = 0.1          #this cannot be zero
    T_list = list(np.linspace(T_min, T_max, num=temp_iterations) )


    data_min = round((1 - alpha) * len(iteration))
    data_max = round((len(iteration)))
    #calculate the upper and lower bound of the sampling range

    #specify the maximum and minimum temperature and construct the list of␣
↪temperature

    for d in range(4):
        dim = 2**(d+2)
        print(dim)
        T_values = []


        for i in range(len(T_list)):

            T = T_list[i]
            #get the temperature from pre-constructed list

            m = 0
            E = 0
            #variable 'm', 'E' are used to sum all the data point m, E value

            j = 0
            #preset the samplerate counter
            k = 0
            #counter for the number of data point used to calculate the mean␣
↪value



            Pro4 = np.exp(-4/T)
            Pro8 = np.exp(-8/T)
            #these two are the pobabilities of fliping the element when dE is 4␣
↪and 8



            lattice = generateLattice(dim)
            #generate Lattice for each temperature
```

```
            for i in iteration:
                x = randrange(0, dim)
                y = randrange(0, dim)
                target = (-1) * lattice[x, y]

                dEhalf = - nsum(x, y, lattice) * target
                #as when '-1' change to '1', the change of E is alway twice of␣
↪the neighboring sum.

                if dEhalf <= 0:
                    lattice[x, y] = target
                elif dEhalf == 2:
                    if Pro4 > random():
                        lattice[x, y] = target
                        #when dE/2 equal to 2, use the probability of dE equal␣
↪to 4, which is Pro4
                elif dEhalf == 4:
                    if Pro8 > random():
                        lattice[x, y] = target
                        #when dE/2 equal to 4, use the probability of dE equal␣
↪to 8, which is Pro8


                if i > data_min and i < data_max:
                    j += 1
                    #bump up the samplerate counter
                    if j%samplerate==0:
                        #every 'samplerate' amout of iteration, collect the data
                        m += magnetization(lattice, dim)
                        E += energy(lattice, dim)
                        k += 1
                        #bump up the number of sample counter


            meanE = E/k
            meanm = m/k
            #taking average

            print(T)
            T_values.append(T)
            average_E[d].append(meanE)
            average_m[d].append(abs(meanm))
            #using the absolute value of m as it is expected to converge to␣
↪either 1 or -1
```

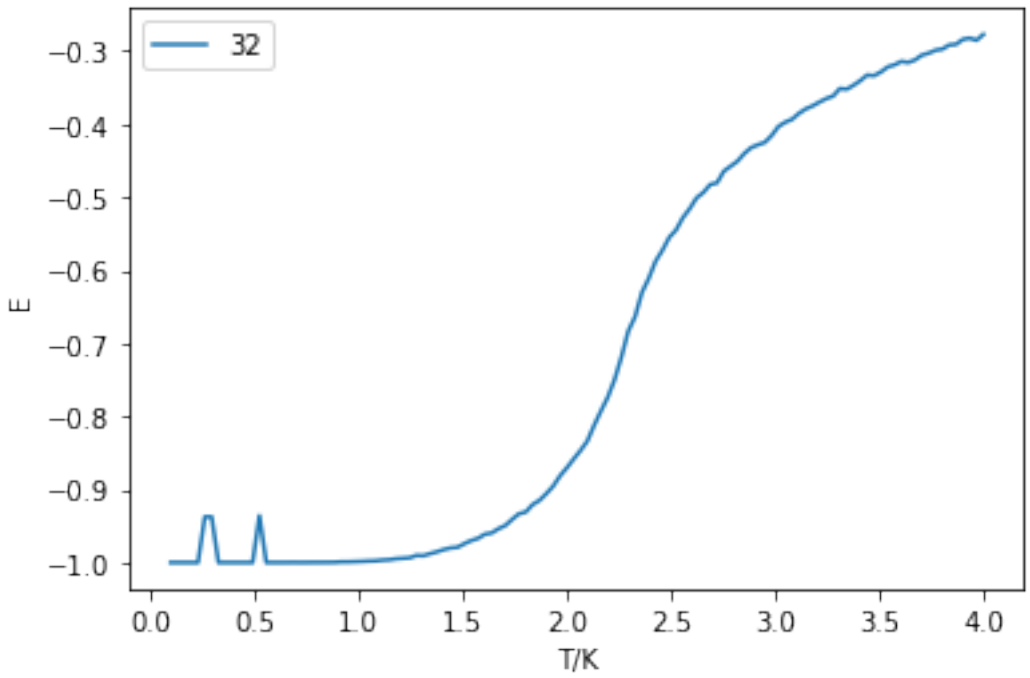
```
            #plt.plot(iteration, m)
            #plt.show()
            #plt.plot(iteration, E, color='red')
            #plt.show()

        try:
            rescale_E[d] = average_E[d]
            for i in range(len(average_E[d])):
                rescale_E[d][i] = float(average_E[d][i])/(2*dim**2)
            #rescale the total energy E with the maximum energy 2*32**2


            print('dim = '+ str(dim))
            print('average E graph')
            plt.figure(0)
            plt.plot(T_values, rescale_E[d], label=str(dim))
            plt.xlabel('T/K')
            plt.ylabel('E')
            plt.legend()



            print('d = '+ str(dim))
            print('average m graph')
            plt.figure(1)
            plt.plot(T_values, average_m[d], label=str(dim))
            plt.legend()
            plt.xlabel('T/K')
            plt.ylabel('m')
        except:
            print('something went wrong')



start = timeit.default_timer()
run()
stop = timeit.default_timer()
#timer
print('Time: ', stop - start)
```

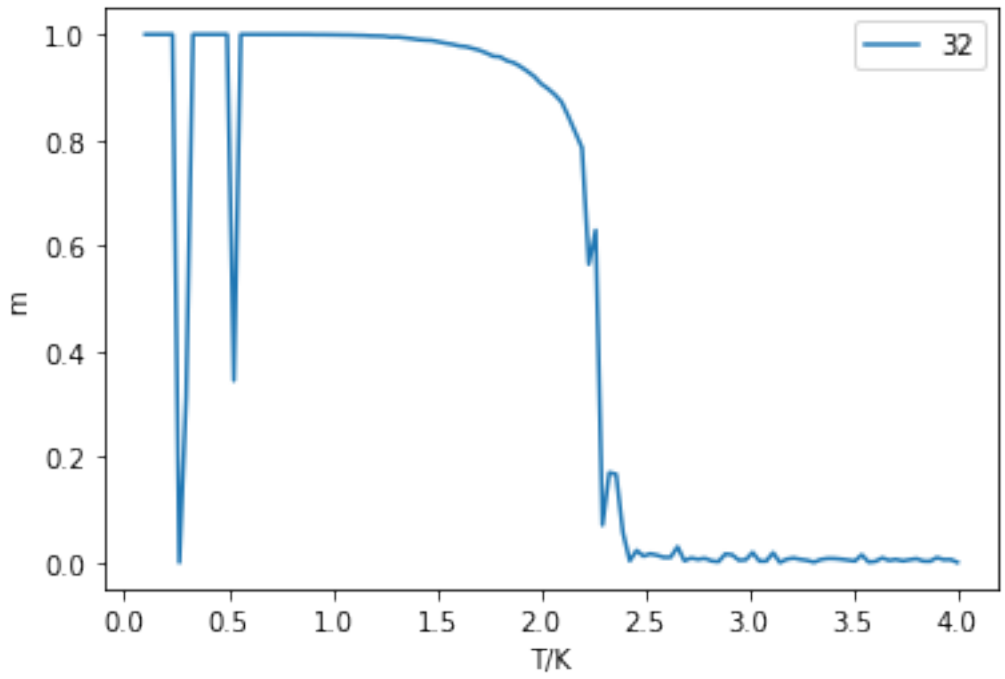
```
32
0.1
0.13277310924369748
0.16554621848739498
0.19831932773109245
```

0.23109243697478993
0.2638655462184874
0.29663865546218493
0.3294117647058824
0.3621848739495799
0.39495798319327735
0.4277310924369748
0.4605042016806723
0.49327731092436977
0.5260504201680672
0.5588235294117647
0.5915966386554622
0.6243697478991597
0.6571428571428571
0.6899159663865546
0.7226890756302521
0.7554621848739496
0.788235294117647
0.8210084033613445
0.853781512605042
0.8865546218487396
0.919327731092437
0.9521008403361345
0.984873949579832
1.0176470588235296
1.050420168067227
1.0831932773109245
1.1159663865546219
1.1487394957983195
1.181512605042017
1.2142857142857144
1.247058823529412
1.2798319327731094
1.312605042016807
1.3453781512605043
1.378151260504202
1.4109243697478993
1.4436974789915968
1.4764705882352942
1.5092436974789918
1.5420168067226891
1.5747899159663867
1.607563025210084
1.6403361344537817
1.6731092436974793
1.7058823529411766
1.7386554621848742
1.7714285714285716

1.8042016806722692
1.8369747899159665
1.869747899159664
1.9025210084033615
1.935294117647059
1.9680672268907564
2.000840336134454
2.033613445378151
2.066386554621849
2.0991596638655463
2.1319327731092437
2.1647058823529415
2.197478991596639
2.230252100840336
2.263025210084034
2.2957983193277314
2.3285714285714287
2.361344537815126
2.394117647058824
2.4268907563025213
2.4596638655462186
2.492436974789916
2.525210084033614
2.557983193277311
2.5907563025210085
2.623529411764706
2.6563025210084037
2.689075630252101
2.7218487394957984
2.7546218487394962
2.7873949579831936
2.820168067226891
2.8529411764705883
2.885714285714286
2.9184873949579835
2.951260504201681
2.984033613445378
3.016806722689076
3.0495798319327734
3.0823529411764707
3.115126050420168
3.147899159663866
3.1806722689075633
3.2134453781512606
3.2462184873949584
3.278991596638656
3.311764705882353
3.3445378151260505

```
3.3773109243697483
3.4100840336134457
3.442857142857143
3.4756302521008404
3.5084033613445382
3.5411764705882356
3.573949579831933
3.6067226890756303
3.639495798319328
3.6722689075630255
3.705042016806723
3.73781512605042
3.770588235294118
3.8033613445378154
3.8361344537815127
3.8689075630252105
3.901680672268908
3.9344537815126053
3.9672268907563026
4.0
dim = 32
average E graph
d = 32
average m graph
Time:  21801.527390699994
```





[ ]:

[53]:
```
temp_iterations = 120
     #taking more datapoints near the critical temperature.
    #the number of temperature assess in the programme
T_max = 4
T_min = 0.1           #this cannot be zero
T_list = list(np.linspace(T_min, T_max, num=temp_iterations) )

np.savetxt("average_m.csv", np.array(average_m[0]), delimiter=",", fmt='%s',␣
 ↪header='header')
np.savetxt("rescale_E.csv", np.array(rescale_E[0]), delimiter=",", fmt='%s',␣
 ↪header='header')
np.savetxt("average_E.csv", np.array(average_E[0]), delimiter=",", fmt='%s',␣
 ↪header='header')
np.savetxt("T.csv", np.array(T_list), delimiter=",", fmt='%s', header='header')
```

[ ]: