Simulation of 2D Ising Model

January 22, 2023

1 Introduction

1.1 The Ising Model

The Ising model is a mathematical model used to describe the behavior of a system of interacting spins. The two-dimensional Ising model (2D Ising model) is a specific case of the Ising model, where the spins are arranged on a 2D lattice. The Ising model is named after the physicist Ernst Ising, who first introduced it in his PhD thesis in 1925. The 2D Ising model is one of the simplest models of a phase transition and has been used to study a wide range of phenomena in physics, materials science, and computer science.

The 2D Ising model is a fundamental model in statistical mechanics and has been used to study a wide range of phenomena, such as critical phenomena, phase transitions, and the behavior of magnets. It is also a challenging problem for computational methods and has been used to test and develop new algorithms for simulating complex systems.

1.2 Use of MCMC for solving Ising Model

Markov Chain Monte Carlo (MCMC) methods, such as the Metropolis-Hastings algorithm, are often used to generate samples from the target probability distribution defined by the Ising model. MCMC methods are useful for simulating the 2D Ising model because they can handle complex systems with many interacting particles and can efficiently explore the state space of the system.

2 Objective of the Project

There are two major objectives of this project and they are

```
i. To simulate the interaction of particles at different
temperatures to find the critical temperature
ii. To simulate the interaction of particles at different
temeprature and see how they behave with respect to time
```

3 Working Methodology

3.1 The Ising Hamiltonian

Hamiltonian function (Hamiltonian) of a system specifies its total energy. It is given by

$$\mathcal{H} = -J \sum_{\langle ij \rangle} S_i S_j.$$

- The spins S_i can take values ± 1,
- $\langle ij \rangle$ implies nearest-neighbor interaction only,
- $J > 0$ is the strength of exchange interaction.

3.2 The Algorithm Used

The Monte Carlo Simulation of 2D Ising model according to Metropolis-Hastings algorithm. The major steps of the algorithm are

1. Generate a initial spin configuration randomly. Use +1 as sping up and -1 as spin down
2. Choose a spin at random
3. Calculate energy (E_old)
4. Flip the spin
5. Calculate new energy (E_new)
6. If (E_new<E_old) accept the flip and go to step 2 for another step else go to step 7
7. Find en = ((E_new-Eold)/T) and generate random number between 0 and 1 say (r)
8. Accept the flip if r>en else do not flip
9. Go to step 2 (Until Converges)

3.3 Tools Used

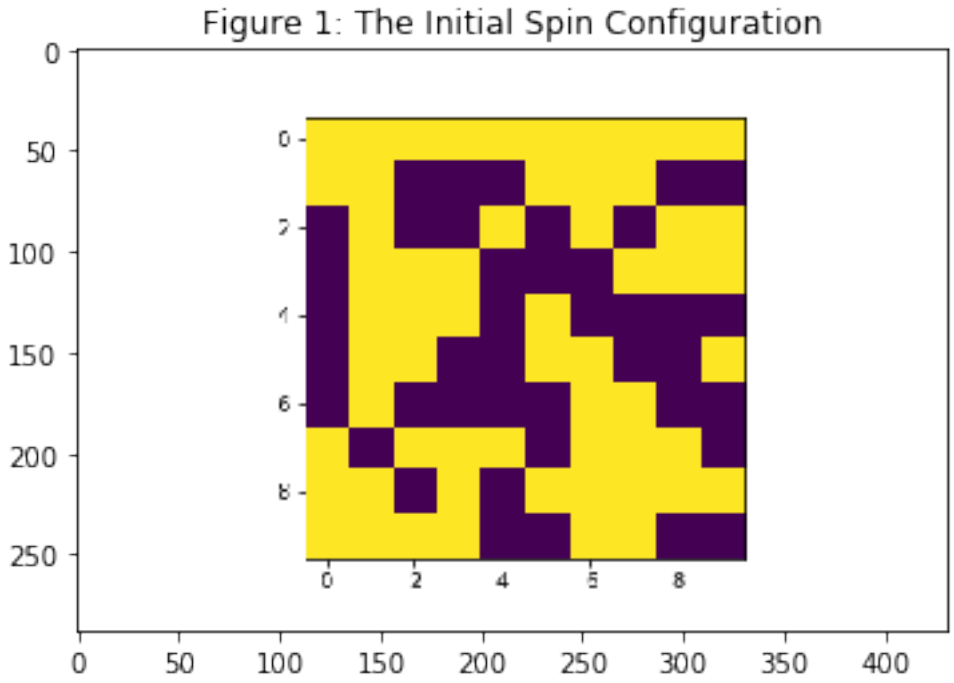
For the implementation of the algorithm following tools are used

1. Python programming Language: Language of choice for implementing the algorithm
2. Numpy
3. Matplotlib
4. Jupyter Notebook

4 Result and Discussion

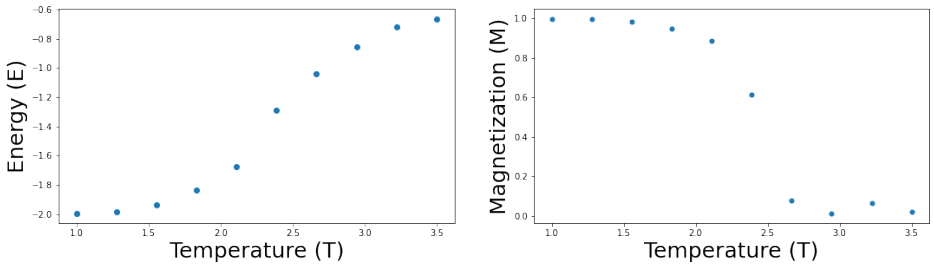
4.1 The Initial Configuration

As mentoined in the algorithm, an initial spin configuration `10x10` lattice is created. The Figure 1 shows the intial configuration



4.2 Result after simulation

After simulating 5 temperature points, energy and magnetization are plotted against temperature. The result is shown in figure 2.

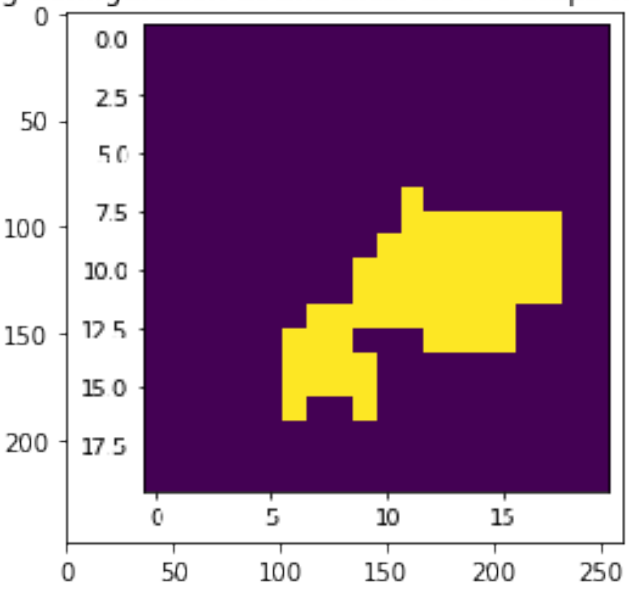


In Figure 2, we can see that there is a sharp change in magnetization between 2.0 and 2.5, which

is near the theoretical critical temperature around 2.26

The spin configuration after 10000 steps for `temperature=1` is given in figure 4

Figure 4: Sping Configuration at End after 10000 steps for temperature=1



The spin configuration at different time for `temperature=2` is given in figure 5

Figure 4: Snapshot of Spin Configuration temperature=2 at different time of 10x10 lattice

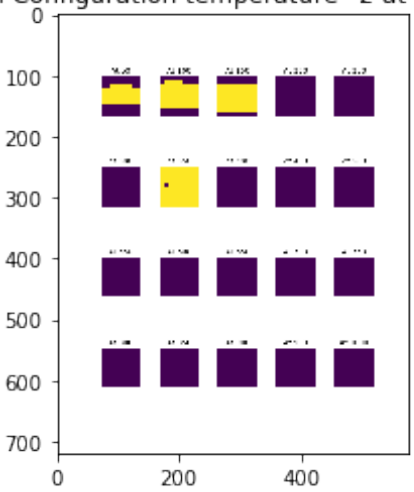
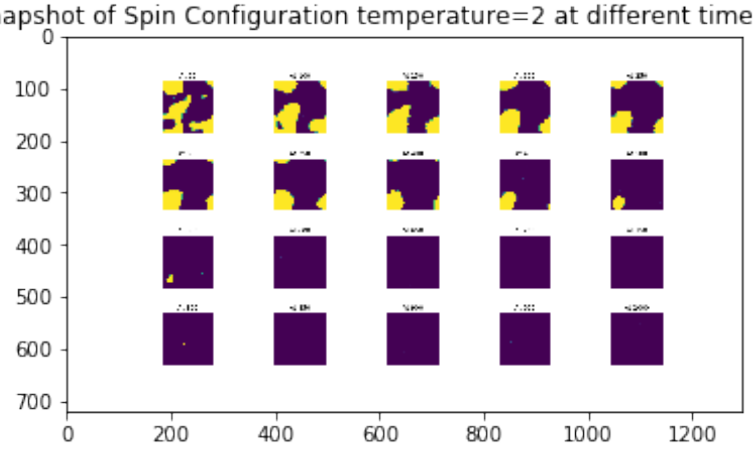


Figure 5: Snapshot of Spin Configuration temperature=2 at different time of 64x64 lattice



From figure 4 we can see that after 1000 iterations the snapshot has the same color which means that the spins are aligned in the same spin configuration. Figure 5 also shows the same thing only difference is the spin configuration. In figure 4 10x10 lattice size is used and in figure 5 64x64 lattice size is used

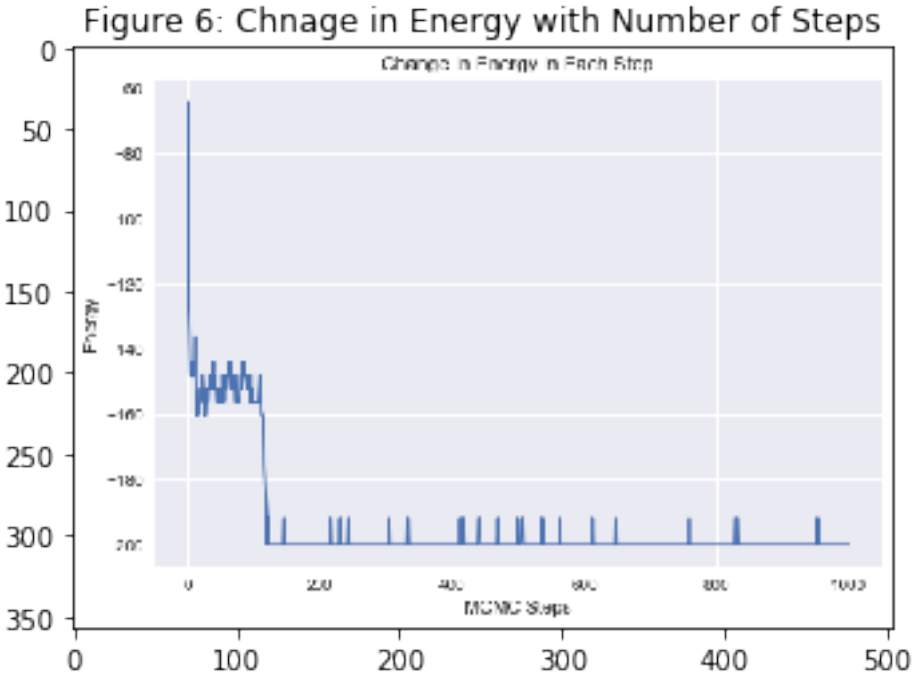


Figure 6 shows the change in energy with the number of steps. From the plot, we can see that energy decreases as the number of steps increases. This is expected as a system always moves from a higher energy state to a lower energy state. After some iteration, there is no significant change in energy which means that the system is in equilibrium.

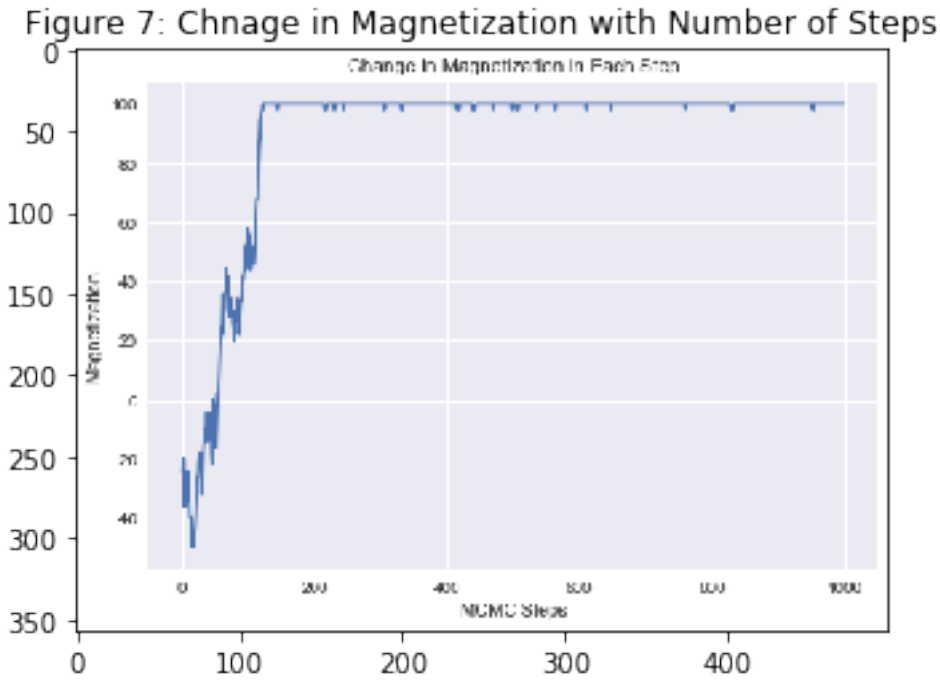


Figure 6 shows the change in magnetization with the number of steps. From the plot, we can see that magnetization increases as the number of steps increases. This is expected as a system is aligning in the same spin configuration. After some iteration, there is no significant change in magnetization which means that the system is in equilibrium.

5 Conclusion

In this project, Metropolish algorithm is sucessfully implemented for solving 2D Ising Model, also the critical temperature for given initial spin configuration is calculated.

6 References

- Hammel, B. D. (2017, Jun 10). B. D. Hammel. Retrieved from http://www.bdhammel.com/ising-model/
- Leban, J. (2020, May 8). Medium. Retrieved from Towards Data Science: https://towardsdatascience.com/monte-carlo-method-applied-on-a-2d-binary-alloy-using-an-ising-model-on-python-70afa03b172b
- Mahapatra, J. M. (2018). A comparative study of 2d Ising model at different boundary conditions using Cellular Automata. International Journal of Modern Physics C.
- Singh, R. (2022, Feburary). Retrieved from https://rajeshrinet.github.io/blog/2014/ising-model/

7 Appendix

The Code Used for this project is "‘ import numpy as np from numpy.random import rand import matplotlib.pyplot as plt from scipy.sparse import spdiags,linalg,eye

def init_state(grid_size=10): """ This Function intalizes the state for random configuration """ spins = np.random.choice([-1, 1], size=(grid_size, grid_size)) # random initial configuration of spins return spins

def calcEnergy(config, grid_size): "’ Energy of a given configuration "’ energy = 0

```
for i in range(len(config)):
    for j in range(len(config)):
        S = config[i, j]
        nb = config[(i + 1) % grid_size, j] + config[i, (j + 1) % grid_size] + config[(i - 1) %
             config[i, (j - 1) % grid_size]
        energy += -nb * S
return energy / 2.  # to compensate for over-counting
```

def calc_magnetization(spins): """ Calculates the Magnetization of given configuration """ return np.sum(spins)

def perform_simulation(spins, beta): """ Run the Simulation """ # Run the Metropolis-Hastings algorithm for i in range(grid_size): for j in range(grid_size): # Choose a random spin a = np.random.randint(0, grid_size) b = np.random.randint(0, grid_size) # Get the spins based on generated random numbers s = spins[a, b] # Compute the Energy nb = spins[(a + 1) % grid_size, b] + spins[a, (b + 1) % grid_size] + spins[(a - 1) % grid_size, b] + spins[a, (b - 1) % grid_size] cost = 2 * s * nb if cost < 0: s = *-1 elif rand() < np.exp(-cost* beta): s *= -1 spins[a, b] = s return spins

n_steps=1000 temperature=1 grid_size=10 init_config_state=init_state(grid_size=grid_size)

plt.imshow(init_config) plt.savefig(‘initial.png’) plt.show()

8 Define the grid size and the number of steps

grid_size = 10 n_steps = 1000

9 Define the range of temperatures to simulate

temperature_range = np.linspace(1, 3, 5)

10 Initialize lists to store the energy and magnetization

energy_list = [] magnetization_list = []

num_temp = 10 # Number of Temperature Points grid_size_val=10 # Size of the Lattice eqSteps = 2**8**
mcSteps = 29
T = np.linspace(1.00, 3.5, num_temp) E,M,C,X = np.zeros(num_temp), np.zeros(num_temp),

np.zeros(num_temp), np.zeros(num_temp) n1, n2 = 1.0/(mcSteps*grid_size*grid_size), 1.0/(mcSteps*mcSteps*grid_size*grid_size)

for temper in range(num_temp): spins = init_state(grid_size=grid_size) E1 = M1 = E2 = M2 = 0 iT=1.0/T[temper] iT2=iT*iT

```
for i in range(eqSteps):
    perform_simulation(spins=spins,beta=iT)
for i in range(mcSteps):
    perform_simulation(spins,iT)
    Ene = calcEnergy(spins,grid_size=grid_size_val)     # calculate the energy
    Mag = calc_magnetization(spins)        # calculate the magnetisation

    E1 = E1 + Ene
    M1 = M1 + Mag
    M2 = M2 + Mag*Mag
    E2 = E2 + Ene*Ene
    # divide by number of sites and iteractions to obtain intensive values
E[temper] = n1*E1
M[temper] = n1*M1
C[temper] = (n1*E2 - n2*E1*E1)*iT2
X[temper] = (n1*M2 - n2*M1*M1)*iT
```

f = plt.figure(figsize=(18, 10)) sp = f.add_subplot(2, 2, 1); plt.scatter(T, E) plt.xlabel("Temperature (T)", fontsize=25) plt.ylabel("Energy (E)", fontsize=25); plt.axis('tight');

sp = f.add_subplot(2, 2, 2); plt.scatter(T, abs(M), s=25) plt.xlabel("Temperature (T)", fontsize=25); plt.ylabel("Magnetization (M)", fontsize=25) plt.axis('tight')

plt.savefig('eng_mag.png') plt.show()

import matplotlib.gridspec as gridspec grid_size = 10 mc_steps=1000

#Set a Temperature temper=1.0

11 Create a counter variable

count = 0

12 Initialize a list to store the spin configurations at different times

spins_list = []

spins = init_state(grid_size) for step in range(mc_steps): spins = perform_simulation(spins=spins,beta=1.0) count=count+1 if count % 50 == 0: spins_list.append(spins.copy())

13 Set the number of rows and columns for the grid

n_rows = 4 n_cols = 5

14 Create a grid of subplots

fig, axes = plt.subplots(n_rows, n_cols, figsize=(8,10))

15 Flatten the axes array

axes = axes.ravel()

16 Iterate over the spin configurations

for i, spins in enumerate(spins_list): axes[i].imshow(spins) axes[i].set_title(f'At {(i*50)+50}') # remove the axis labels and ticks axes[i].axis('off')

plt.subplots_adjust(wspace=0.5, hspace=0.5) plt.savefig('multi.png') plt.show()

import matplotlib.gridspec as gridspec grid_size = 64 mc_steps=1000

#Set a Temperature temper=1.0

17 Create a counter variable

count = 0

18 Initialize a list to store the spin configurations at different times

spins_list = []

spins = init_state(grid_size) for step in range(mc_steps): spins = perform_simulation(spins=spins,beta=1.0) count=count+1 if count % 50 == 0: spins_list.append(spins.copy())

Set the number of rows and columns for the grid n_rows = 4 n_cols = 5

19 Create a grid of subplots

fig, axes = plt.subplots(n_rows, n_cols, figsize=(18,10))

20 Flatten the axes array

axes = axes.ravel()

21 Iterate over the spin configurations

for i, spins in enumerate(spins_list): axes[i].imshow(spins) axes[i].set_title(f'At {(i*50)+50}') # remove the axis labels and ticks axes[i].axis('off')

plt.subplots_adjust(wspace=0.5, hspace=0.5) plt.savefig('multi_1.png') plt.show()

mc_steps = 1000 #Set a Temperature temper=1.0

22 Create a counter variable

count = 0

23 Initialize a list to store the spin configurations at different times

spins_list = [] eng_list = [] mag_list = [] spins = init_state(grid_size) for step in range(mc_steps): spins = perform_simulation(spins=spins,beta=1.0) spins_list.append(spins.copy()) eng_list.append(calcEnergy(spins,10)) mag_list.append(calc_magnetization(spins))

plt.style.use('seaborn') plt.plot(range(mc_steps),eng_list) plt.title('Change in Energy in Each Step') plt.ylabel('Energy') plt.xlabel('MCMC Steps')

plt.style.use('seaborn') plt.plot(range(mc_steps),mag_list) plt.title('Change in Magnetization in Each Step') plt.ylabel('Magnetization') plt.xlabel('MCMC Steps')