Homework 01: Monte Carlo Simulation of the Ising Model

Katha Haldar

Department of Physics, Indiana University, Indianapolis, IN 46202

Introduction

The Ising model is one of the simplest and most extensively studied models in statistical physics. Originally introduced to model ferromagnetism, the Ising model captures the essential physics of systems with spin-like interactions. In its two-dimensional form, it consists of a square lattice where each lattice site contains a spin variable, S_i, that can take values of $+1$ or -1. The spins interact with their nearest neighbors, and the system's energy is governed by the Hamiltonian:

$$H = -J \sum_{\langle i,j \rangle} S_i S_j - H \sum_i S_i,$$

where J is the interaction strength, H represents an external magnetic field, and $\langle i, j \rangle$ denotes summation over all nearest-neighbor pairs.

Monte Carlo simulations, particularly the Metropolis algorithm, provide an efficient way to study the statistical properties of the Ising model at equilibrium. By generating spin configurations according to their Boltzmann weights, the algorithm enables the calculation of observables like energy, magnetization, and susceptibility over a range of temperatures.

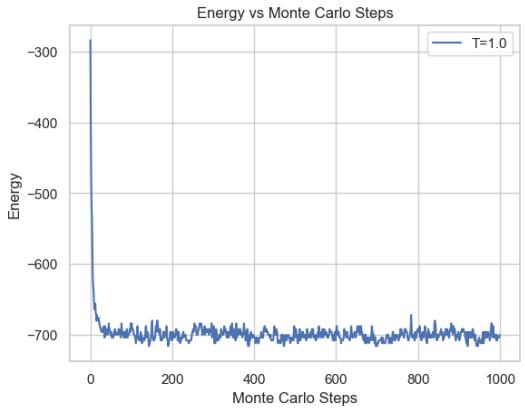
Python Code: Energy *vs.* Monte Carlo Steps Simulation

The following Python code simulates the energy vs temperature behavior for the 2D Ising model using Monte Carlo methods:

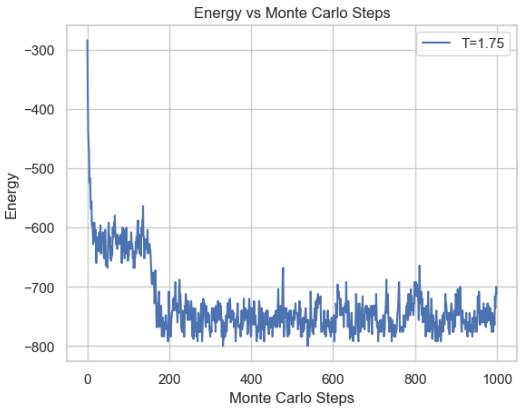
```
import numpy as np
import matplotlib.pyplot as plt
```

```
from tqdm import tqdm
from ising_model import IsingModel2D  # Assuming IsingModel2D is
                                      implemented in this module
def main():
    # Simulation parameters
    L = 20           # Lattice size (20x20)
    J = 1            # Interaction energy
    H = 0            # External magnetic field
    steps = 1000     # Number of Monte Carlo steps per temperature
    temperatures = np.linspace(1.0, 4.0, 5)  # Temperature range
                                             from 1.0 to 4.0 (5 sample
                                             points)
    average_energies = []
    # Initialize progress bar
    print("Simulating Energy vs Monte Carlo Steps for 2D Ising Model
                                             :")
    for T in tqdm(temperatures):
        # Initialize the model
        model = IsingModel2D(L, T, J, H)
        # Perform simulation and get energies over steps
        energies = model.simulate(steps, equilibration=False)
        # Calculate average energy per spin
        avg_energy = np.mean(energies) / (L**2)
        average_energies.append(avg_energy)
        # Plot energy vs Monte Carlo steps for the current
                                               temperature
        model.plot_energy(energies, T)
    # Plot Energy vs Temperature
    plt.figure(figsize=(8,6))
    plt.plot(temperatures, average_energies, 'o-', color='blue')
    plt.xlabel('Temperature (T)')
    plt.ylabel('Average Energy per Spin <E>')
    plt.title('Energy vs Temperature for 2D Ising Model')
    plt.grid(True)
    plt.show()
if __name__ == "__main__":
    main()
```

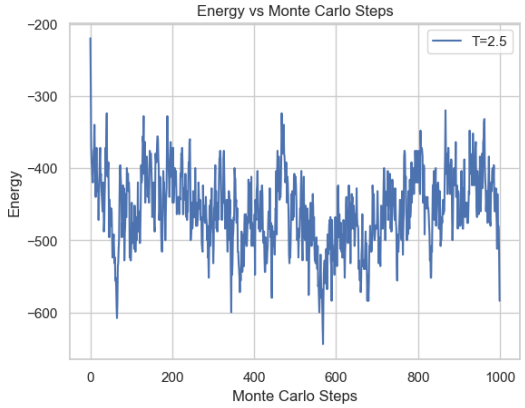
Results



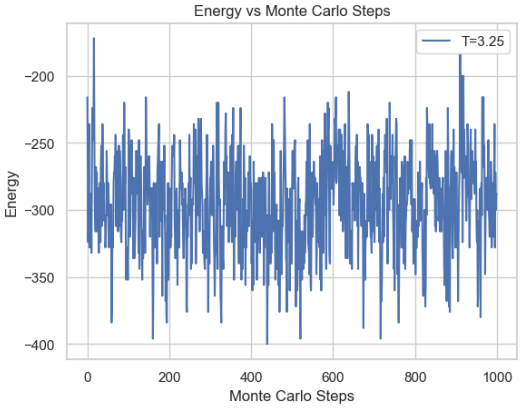
(a) T = 1.0



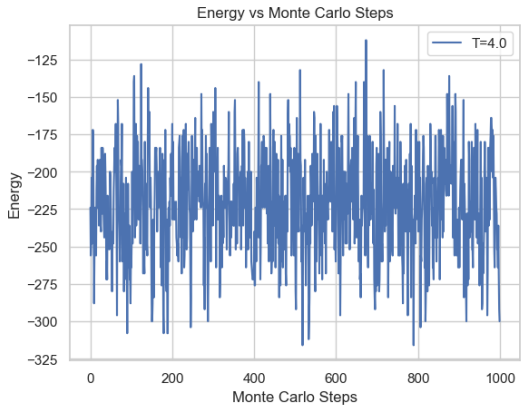
(b) T = 1.5



(c) T = 2.0



(d) T = 2.5



(e) T = 3.0

Figure 1: Energy vs Monte Carlo Steps for different temperatures in the 2D Ising Model. Each subplot shows the energy variation with Monte Carlo steps for a specific temperature.

Python Code: Configurations for $T = 0.4$

The following Python code simulates the energy vs temperature behavior for the 2D Ising model using Monte Carlo methods:

```
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt

class Ising():
    ''' Simulating the Ising model '''

    ## Monte Carlo moves
    def mcmove(self, config, N, beta):
        '''
        Execute the Monte Carlo moves using the Metropolis algorithm
        to satisfy the detailed balance condition.
        '''
        for i in range(N):
            for j in range(N):
                a = np.random.randint(0, N)
                b = np.random.randint(0, N)
                s = config[a, b]
                nb = config[(a+1)%N, b] + config[a, (b+1)%N] +
                                                    config[(a-1)%N, b]
                                                     + config[a, (b-1)
                                                    %N]
                cost = 2 * s * nb
                if cost < 0:
                    s *= -1
                elif rand() < np.exp(-cost * beta):
                    s *= -1
                config[a, b] = s
        return config

    def simulate(self):
        ''' Simulate the Ising model '''
        N, temp = 64, 0.4        # Initialize the lattice size and
                                                temperature
        config = 2 * np.random.randint(2, size=(N, N)) - 1
        f = plt.figure(figsize=(15, 15), dpi=80)
        self.configPlot(f, config, 0, N, 1)
```

```
        msrmnt = 1001
        for i in range(msrmnt):
            self.mcmove(config, N, 1.0 / temp)
            if i == 1:       self.configPlot(f, config, i, N, 2,
                                             cmap = 'plasma')
            if i == 5:       self.configPlot(f, config, i, N, 3,
                                             cmap = 'plasma')
            if i == 50:      self.configPlot(f, config, i, N, 4,
                                             cmap = 'plasma')
            if i == 100:     self.configPlot(f, config, i, N, 5,
                                             cmap = 'plasma')
            if i == 1000:    self.configPlot(f, config, i, N, 6,
                                             cmap = 'plasma')

    def configPlot(self, f, config, i, N, n_, cmap = 'plasma'):
        ''' Plot the configuration at a given time step '''
        X, Y = np.meshgrid(range(N), range(N))
        sp = f.add_subplot(3, 3, n_)
        plt.setp(sp.get_yticklabels(), visible=False)
        plt.setp(sp.get_xticklabels(), visible=False)
        sp.pcolormesh(X, Y, config, cmap=plt.cm.plasma)
        sp.set_title(f'Time={i}')
        sp.axis('tight')

# Instantiate the Ising class and run the simulation
if __name__ == "__main__":
    model = Ising()
    model.simulate()
    plt.show()
```

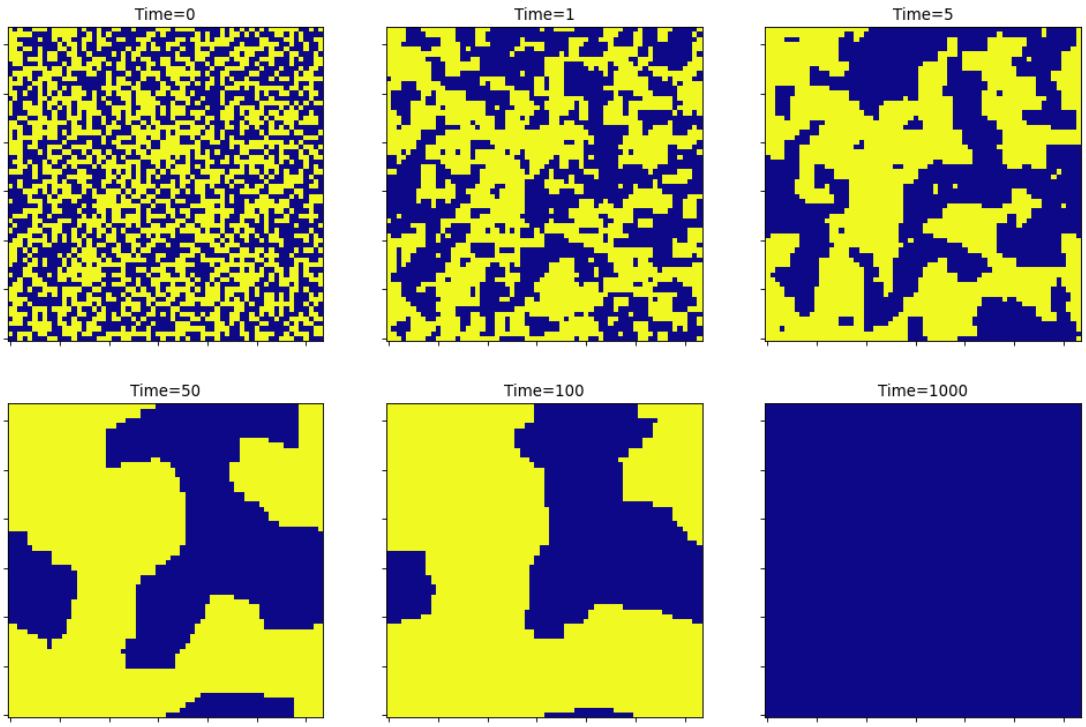


Figure 2: Domain Growth in Monte Carlo Simulation

Python Code: Main Simulation

The following Python code simulates the energy vs temperature behavior for the 2D Ising model using Monte Carlo methods:

```
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm  # For displaying progress bars

class IsingModel2D:
    def __init__(self, L, T, J=1, H=0):
        self.L = L
        self.T = T
        self.J = J
        self.H = H
        self.spins = np.random.choice([-1, 1], size=(L, L))
        self.energy = self.calculate_total_energy()
```

```
    def calculate_total_energy(self):
        energy = 0
        for i in range(self.L):
            for j in range(self.L):
                S = self.spins[i, j]
                neighbors = self.spins[(i+1) % self.L, j] + \
                            self.spins[i, (j+1) % self.L] + \
                            self.spins[(i-1) % self.L, j] + \
                            self.spins[i, (j-1) % self.L]
                energy += -self.J * S * neighbors
        energy /= 2
        energy -= self.H * np.sum(self.spins)
        return energy

    def delta_energy(self, i, j):
        S = self.spins[i, j]
        neighbors = self.spins[(i+1) % self.L, j] + \
                    self.spins[i, (j+1) % self.L] + \
                    self.spins[(i-1) % self.L, j] + \
                    self.spins[i, (j-1) % self.L]
        delta_E = 2 * S * (self.J * neighbors + self.H)
        return delta_E

    def metropolis_step(self):
        for _ in range(self.L**2):
            i = np.random.randint(0, self.L)
            j = np.random.randint(0, self.L)
            dE = self.delta_energy(i, j)
            if dE < 0 or np.random.rand() < np.exp(-dE / self.T):
                self.spins[i, j] *= -1
                self.energy += dE

    def simulate(self, steps, equilibration=True):
        energies = []
        magnetizations = []
        for step in range(steps):
            self.metropolis_step()
            if not equilibration or step >= steps // 2:
                energies.append(self.energy)
                magnetizations.append(np.abs(np.sum(self.spins)) / (
                                                    self.L ** 2))
        return energies, magnetizations
```

```
def main():
    L = 20
    J = 1
    H = 0
    steps = 2000
    temperatures = np.linspace(1.0, 4.0, 20)
    average_energies = []
    average_magnetizations = []

    # Initialize lists to store specific heats and susceptibilities
    specific_heats = []
    susceptibilities = []

    # Initialize progress bar
    print("Simulating Energy vs Temperature for 2D Ising Model:")
    for T in tqdm(temperatures):
        model = IsingModel2D(L, T, J, H)
        energies, magnetizations = model.simulate(steps,
                                                  equilibration=True)
        avg_energy = np.mean(energies)
        avg_magnetization = np.mean(magnetizations)

        # Store averages
        average_energies.append(avg_energy)
        average_magnetizations.append(avg_magnetization)

        # Calculate specific heat and susceptibility
        energy_fluctuation = np.var(energies)
        specific_heat = energy_fluctuation / (L**2 * T**2)
        specific_heats.append(specific_heat)  # Store specific heat

        magnetization_fluctuation = np.var(magnetizations)
        susceptibility = magnetization_fluctuation / (L**2 * T)
        susceptibilities.append(susceptibility)  # Store
                                                 susceptibility

    # Plot Energy vs Temperature
    plt.figure(figsize=(14, 10))

    # Energy vs Temperature
    plt.subplot(2, 2, 1)
```

```
plt.plot(temperatures, average_energies, 'o-', color='royalblue'
                                    , markersize=8, label='<E>')
plt.xlabel('Temperature (T)', fontsize=18)
plt.ylabel('Average Energy per Spin <E>', fontsize=18)
plt.title('Energy vs Temperature', fontsize=20)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(fontsize=14)

# Magnetization vs Temperature
plt.subplot(2, 2, 2)
plt.plot(temperatures, average_magnetizations, 'o-', color='
                                    darkorange', markersize=8,
                                    label='<M>')
plt.xlabel('Temperature (T)', fontsize=18)
plt.ylabel('Magnetization', fontsize=18)
plt.title('Magnetization vs Temperature', fontsize=20)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(fontsize=14)

# Specific Heat vs Temperature (Line plot)
plt.subplot(2, 2, 3)
plt.plot(temperatures, specific_heats, color='indianred',
                                    linewidth=2, marker='o',
                                    markersize=8, label='C')
plt.xlabel('Temperature (T)', fontsize=18)
plt.ylabel('Specific Heat (C)', fontsize=18)
plt.title('Specific Heat vs Temperature', fontsize=20)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(fontsize=14)

# Susceptibility vs Temperature (Line plot)
plt.subplot(2, 2, 4)
plt.plot(temperatures, susceptibilities, color='mediumseagreen',
                                     linewidth=2, marker='o',
                                    markersize=8, label='$\Chi$')
plt.xlabel('Temperature (T)', fontsize=18)
plt.ylabel('Susceptibility ($\Ch$)', fontsize=18)
plt.title('Susceptibility vs Temperature', fontsize=20)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend(fontsize=14)

plt.tight_layout()
```

```
    plt.savefig('ising_model_results.png', dpi=300)  # Save figure
                                                     with high resolution
    plt.show()

if __name__ == "__main__":
    main()
```

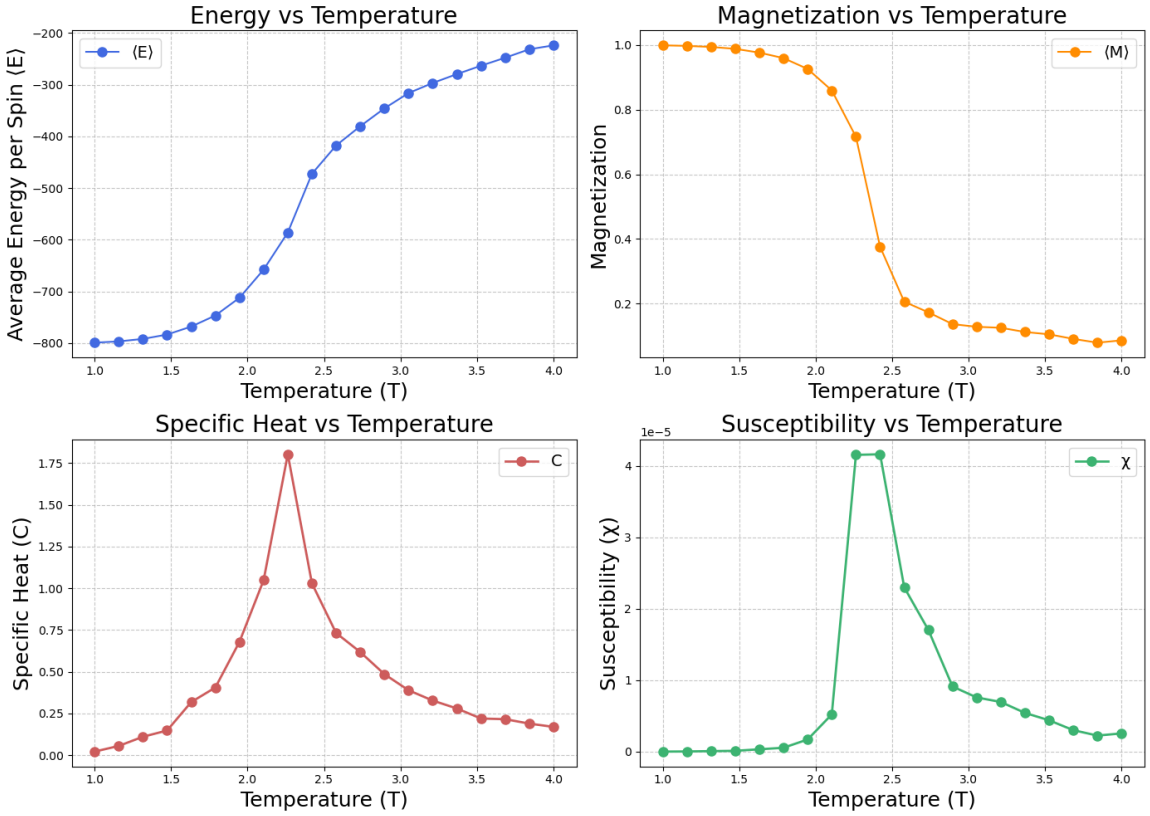


Figure 3: Energy vs Temperature, Magnetization vs Temperature, Specific Heat vs Temperature, and Susceptibility vs Temperature for the 2D Ising Model.

Discussion

The system undergoes a second-order phase transition at the critical temperature T_c. For temperatures less than T_c, the system magnetizes, and the state is called the ferromagnetic or the ordered state. This amounts to a globally ordered state due to the presence of local interactions between the spins. For temperatures greater than

T_c, the system is in the disordered or the paramagnetic state. In this case, there are no long-range correlations between the spins.

The order parameter

$$m = \frac{\langle S \rangle}{N},$$

for this system is the average magnetization. The order parameter distinguishes the two phases realized by the system. It is zero in the disordered state, while non-zero in the ordered, ferromagnetic state.

The one-dimensional (1D) Ising model does not exhibit the phenomenon of a phase transition, while higher dimensions do. This can be argued based on arguments due to Peierls, related to the net change in free energy, $F = E - TS$. Here E and S are, respectively, the energy and entropy of the system. We estimate the net change in free energy for introducing a disorder in an otherwise ordered system. The ordered state can only be stable if the net change in free energy is positive, $\Delta F > 0$, for any non-zero temperature.

Domain Wall and Free Energy in the 1D Ising Model

In the 1D Ising model, introducing a domain wall (defect) in an ordered state leads to an increase in the energy and a change in entropy. Let's analyze the effects step by step.

Energy Increase Due to a Domain Wall

When a domain wall is introduced into an ordered state, the energy increases by $4J$, where J is the interaction strength. This energy increase arises because a domain wall represents a boundary between two regions of opposite spin, which costs energy due to the interaction between spins on either side of the wall.

$$\Delta E = 4J$$

Entropy Change

The introduction of the domain wall also introduces an entropy change. Since there are N possible positions for the domain wall, the entropy change is given by:

$$\Delta S = k_B \ln N$$

where k_B is the Boltzmann constant, and N is the number of possible positions for the domain wall.

Net Change in Free Energy

The net change in the free energy ΔF is given by the difference between the energy increase and the entropy change due to the domain wall:

$$\Delta F = \Delta E - T\Delta S$$

$$\Delta F = 4J - k_B T \ln N$$

As $N \to \infty$, the second term becomes large, and thus the net change in free energy is always negative for $N \to \infty$ (assuming $T > 0$).

Conclusion: No Spontaneous Symmetry Breaking in 1D

Since ΔF is negative as $N \to \infty$, the system favors a disordered state rather than the ordered state with a domain wall. Therefore, there is no spontaneous symmetry breaking in a 1D Ising chain for an infinite system size. This argument can be generalized to any domain of length L in higher dimensions.

Critical Temperature in 2D Ising Model with Defects

The system described involves islands of defects in the 2D Ising model. The key parameters given are:

- The perimeter of each defect island scales as $L = \varepsilon N^2$, where $0 < \varepsilon < 1$.
- The number of defect islands scales as $3\varepsilon N^2$.
- The energy cost for a defect island is $\Delta E = \varepsilon^4 J N^2$, where J is the interaction strength between spins.
- The free energy difference associated with these defects is $\Delta F = \varepsilon^4 J N^2 - k_B T \ln\left(\frac{N^2}{3\varepsilon N^2}\right)$, where k_B is the Boltzmann constant and T is the temperature.

We begin by simplifying the expression for the free energy difference. First, simplify the logarithmic term:

$$\Delta F = \varepsilon^4 J N^2 - k_B T \ln\left(\frac{1}{3\varepsilon}\right)$$

$$\Delta F = \varepsilon^4 J N^2 + k_B T \ln(3\varepsilon)$$

Condition for Defect Formation

For the defects to be present, the system must favor defect formation. This happens when the free energy is minimized. At the critical temperature T_c, the cost of defects due to energy and the thermal fluctuation term should balance. Thus, we set:

$$\varepsilon^4 J N^2 \sim k_B T_c \ln(3\varepsilon)$$

Solving for T_c, we obtain:

$$T_c \sim \frac{\varepsilon^4 J N^2}{k_B \ln(3\varepsilon)}$$

Final Estimate of T_c

For small ε, we can approximate $\ln(3\varepsilon)$ as $\ln(\varepsilon)$. Hence, the critical temperature scales approximately as:

$$T_c \sim \frac{J}{k_B}$$

Conclusion: Thus, the critical temperature T_c for the 2D Ising model with defect islands is primarily determined by the interaction strength J, and it does not depend directly on the system size N. The logarithmic dependence on ε is a secondary effect.