# FYS3150-Project4

# Studies of phase transitions in magnetic systems

Chenghao Liu

November 17, 2018

## 1. Introduction

The goal of this project is to simulate the Ising model, which is a useful model for studying magnetic phase transitions. Here we are simulating a two-dimensional Ising model. The direction of each unit in the Ising model is represented by +1 and -1. With the help of the electromagnetic equation, we can simulate the change of the Ising model.

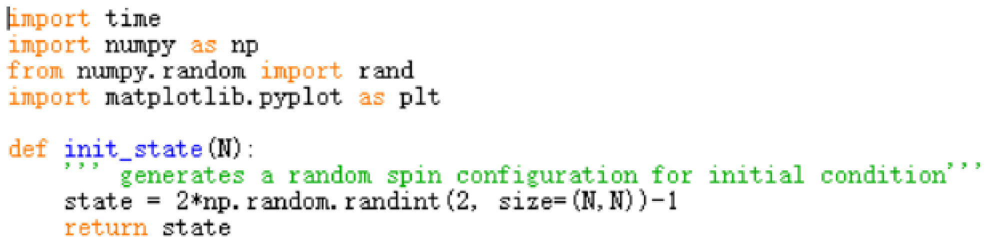
The Ising model is a very popular model and has some applications in many places. In one and two dimensions it has analytical solutions to several expectation values and it gives a qualitatively good understanding of several types of phase transitions.

In Project4, the first task is to let us calculate the Ising model of a 2*2 lattice and find the values of E, M, Cv, and K. The second task is to let us implement an Ising model that can simulate a 2*2 grid by code, and this code can calculate the values of E, M, Cv, and X. Set T to 1, and compare the result with a. The third task is to extend the lattice to 20*20 and calculate the time when the Ising model reaches the equilibrium state. At this time, we can regard Monte Carlo Cycles as the time for reaching equilibrium state. Here we can study the state of its stability and set different T to conduct research. The fourth task is to analyze the probability distribution, that is, the probability distribution of the energy E when the Ising model reaches the equilibrium state, which can be calculated by simply calculating the number of times a given energy appears in the calculation. Once the steady state is reached, the calculation begins. Compare the results with the energy variance and discuss. The fifth task is to simulate different L and calculate its E, M, Cv, K. And simulate E, M, Cv, K under different T, also draw a figure as a function of T. The sixth task is to calculate the situation under critical temperature and find the Tc when L tends to infinity.

## 2. Method

Here I use Python to simulate the Ising model, because Python's numpy and matplotlib can satisfy the simulation of the Ising model. I used the metropolis algorithm and the monte carlo simulation to write the code. The whole code is roughly composed of four parts.

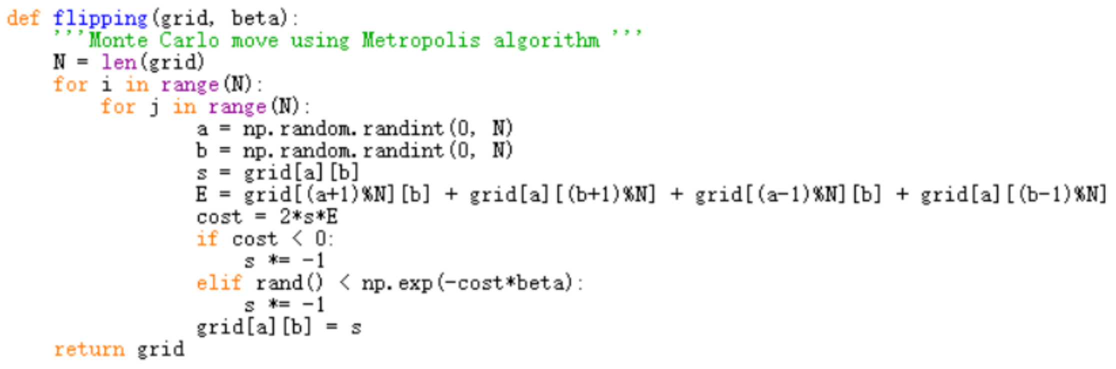
The first part is to generate the initial matrix. The code is as follows



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

def init_state(N):
    ''' generates a random spin configuration for initial condition'''
    state = 2*np.random.randint(2, size=(N,N))-1
    return state
```

Figure 1 Code for part 1

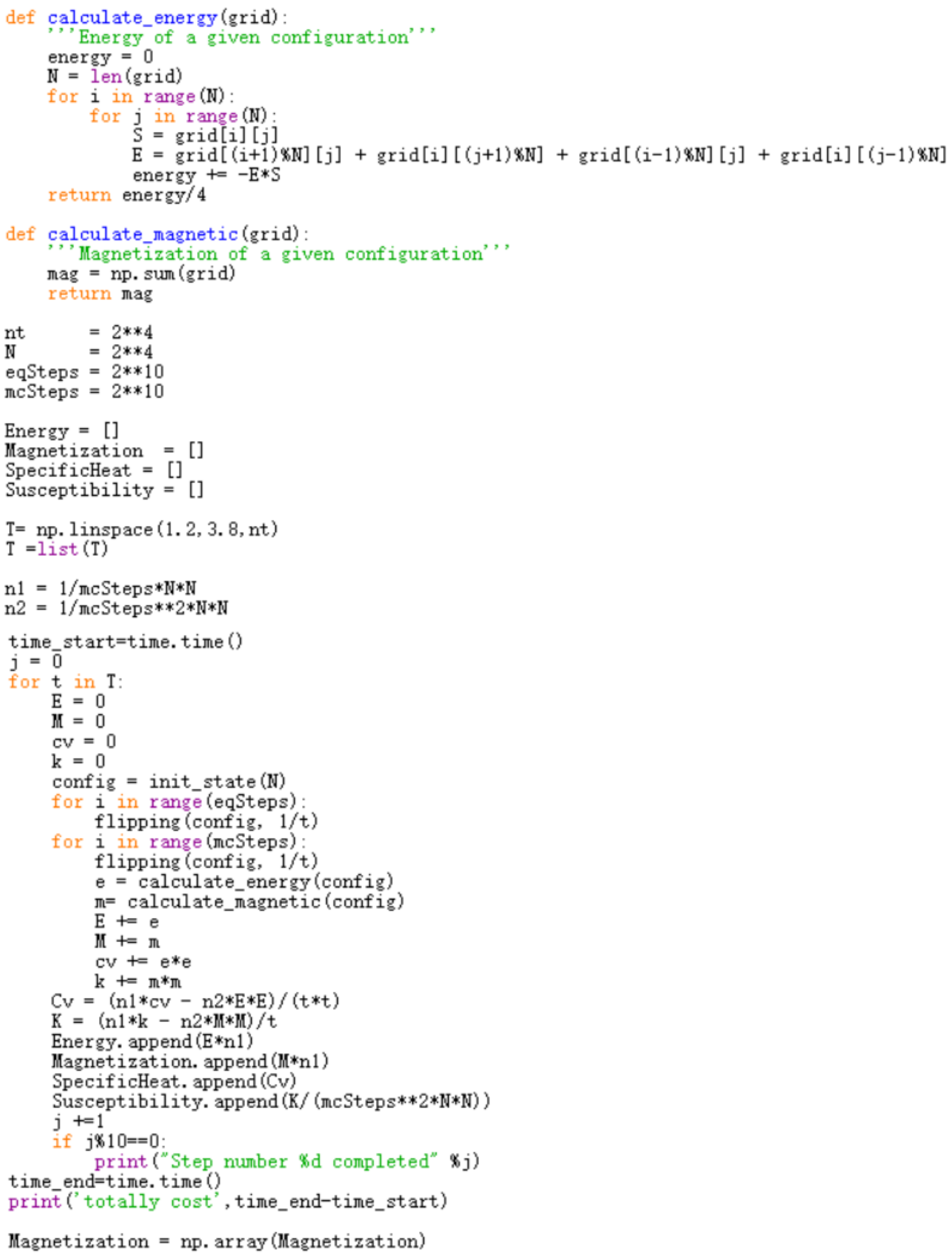
The second part is to make it reach the equilibrium state. The calculation method is to make each pointer rotate randomly. When the energy decreases after rotation, it will accept the flip. If not, it will generate a random number of 0 to 1. If the probability is less than exp(-E /(kT)), then it flips, the code is as follows



```python
def flipping(grid, beta):
    '''Monte Carlo move using Metropolis algorithm '''
    N = len(grid)
    for i in range(N):
        for j in range(N):
                a = np.random.randint(0, N)
                b = np.random.randint(0, N)
                s = grid[a][b]
                E = grid[(a+1)%N][b] + grid[a][(b+1)%N] + grid[(a-1)%N][b] + grid[a][(b-1)%N]
                cost = 2*s*E
                if cost < 0:
                    s *= -1
                elif rand() < np.exp(-cost*beta):
                    s *= -1
                grid[a][b] = s
    return grid
```

Figure 2 Code for part 2

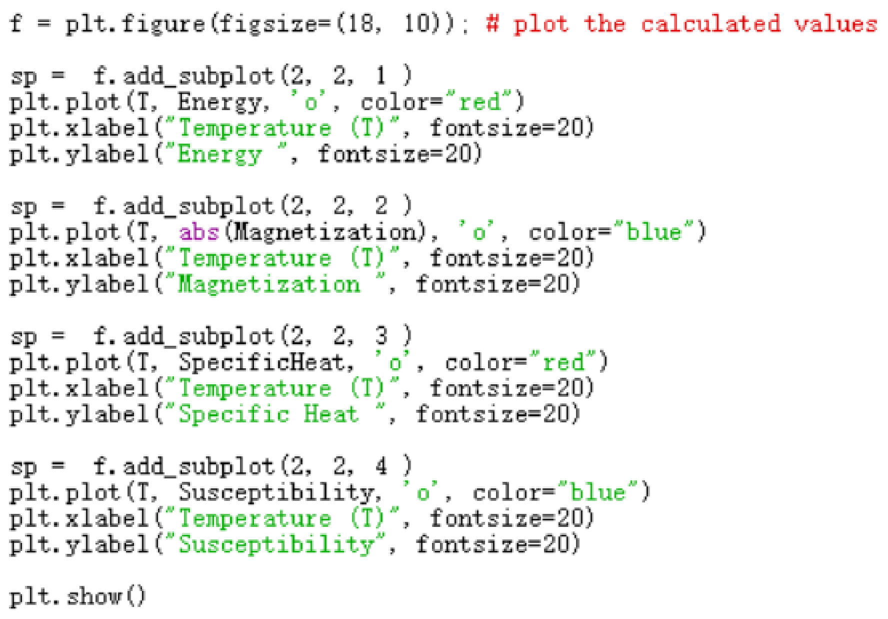
The third part is to calculate the energy E, the magnetic moment M, the specific heat capacity Cv, and the magnetic susceptibility K. The algorithm of the code is to calculate E, M, Cv, K at each temperature after the Ising model is stable. Code shows as below



```python
def calculate_energy(grid):
    '''Energy of a given configuration'''
    energy = 0
    N = len(grid)
    for i in range(N):
        for j in range(N):
            S = grid[i][j]
            E = grid[(i+1)%N][j] + grid[i][(j+1)%N] + grid[(i-1)%N][j] + grid[i][(j-1)%N]
            energy += -E*S
    return energy/4

def calculate_magnetic(grid):
    '''Magnetization of a given configuration'''
    mag = np.sum(grid)
    return mag

nt      = 2**4
N       = 2**4
eqSteps = 2**10
mcSteps = 2**10

Energy = []
Magnetization  = []
SpecificHeat = []
Susceptibility = []

T= np.linspace(1.2,3.8,nt)
T =list(T)

n1 = 1/mcSteps*N*N
n2 = 1/mcSteps**2*N*N

time_start=time.time()
j = 0
for t in T:
    E = 0
    M = 0
    cv = 0
    k = 0
    config = init_state(N)
    for i in range(eqSteps):
        flipping(config, 1/t)
    for i in range(mcSteps):
        flipping(config, 1/t)
        e = calculate_energy(config)
        m= calculate_magnetic(config)
        E += e
        M += m
        cv += e*e
        k += m*m
    Cv = (n1*cv - n2*E*E)/(t*t)
    K = (n1*k - n2*M*M)/t
    Energy.append(E*n1)
    Magnetization.append(M*n1)
    SpecificHeat.append(Cv)
    Susceptibility.append(K/(mcSteps**2*N*N))
    j +=1
    if j%10==0:
        print("Step number %d completed" %j)
time_end=time.time()
print('totally cost',time_end-time_start)

Magnetization = np.array(Magnetization)
```

Figure 3 Code for part 3

The fourth part is drawing, this part is more intuitive and concise, the code is as follows



```
f = plt.figure(figsize=(18, 10)); # plot the calculated values

sp =  f.add_subplot(2, 2, 1 )
plt.plot(T, Energy, 'o', color="red")
plt.xlabel("Temperature (T)", fontsize=20)
plt.ylabel("Energy ", fontsize=20)

sp =  f.add_subplot(2, 2, 2 )
plt.plot(T, abs(Magnetization), 'o', color="blue")
plt.xlabel("Temperature (T)", fontsize=20)
plt.ylabel("Magnetization ", fontsize=20)

sp =  f.add_subplot(2, 2, 3 )
plt.plot(T, SpecificHeat, 'o', color="red")
plt.xlabel("Temperature (T)", fontsize=20)
plt.ylabel("Specific Heat ", fontsize=20)

sp =  f.add_subplot(2, 2, 4 )
plt.plot(T, Susceptibility, 'o', color="blue")
plt.xlabel("Temperature (T)", fontsize=20)
plt.ylabel("Susceptibility", fontsize=20)

plt.show()
```

Figure 4 Code for part 4

For different parts of the project, you can set different N to change the grid point L, you can randomly generate the initial Ising model, or you can set the initial Ising model by setting it yourself. Then add an if function to determine if the Ising model has stabilized. It required many steps for the Monte Carlo cycle to achieve stability, which can be regard as the time required to reach steady state. As the code we have discussed in the above Introduction and Method section, we can extract some of the code to complete the images and answers needed for each problem of the project.

## 3. Implementation

### 4a

The work of this part is to calculate a 2*2 Ising model to find the values of E, M, Cv, K. For 2*2 grids, we only need to consider the inner space between the two grids of each grid neighbor. Therefore, its internal energy can be expressed as

$$E = -J(s_{11}s_{12} + s_{11}s_{21} + s_{22}s_{12} + s_{22}s_{21})$$

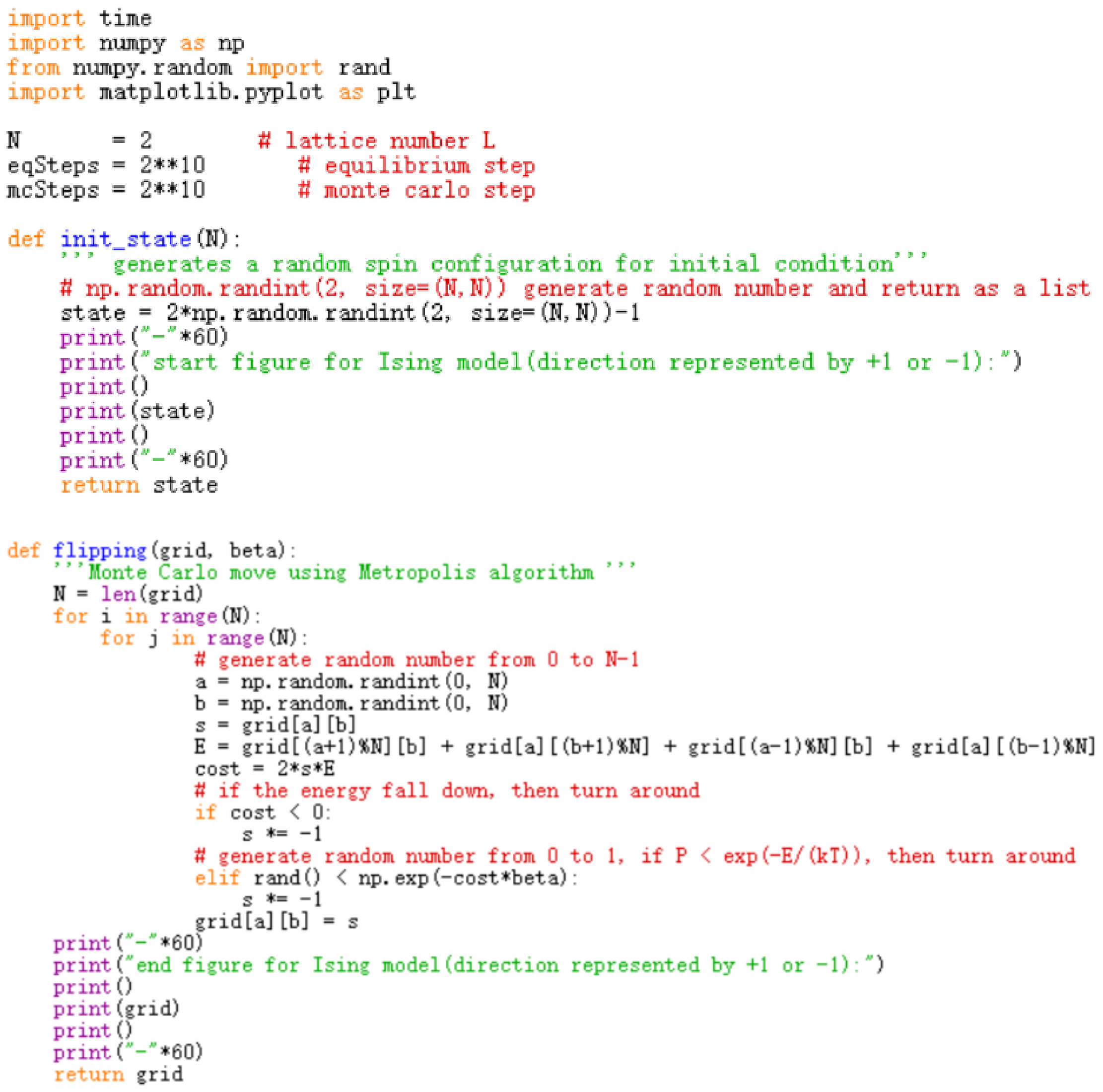
For the magnetic moment, since the direction of our simulation is represented by +1 and -1, the total magnetic moment is The sum of the directions, that is

$$\mathrm{M} = s_{11} + s_{12} + s_{21} + s_{22}$$

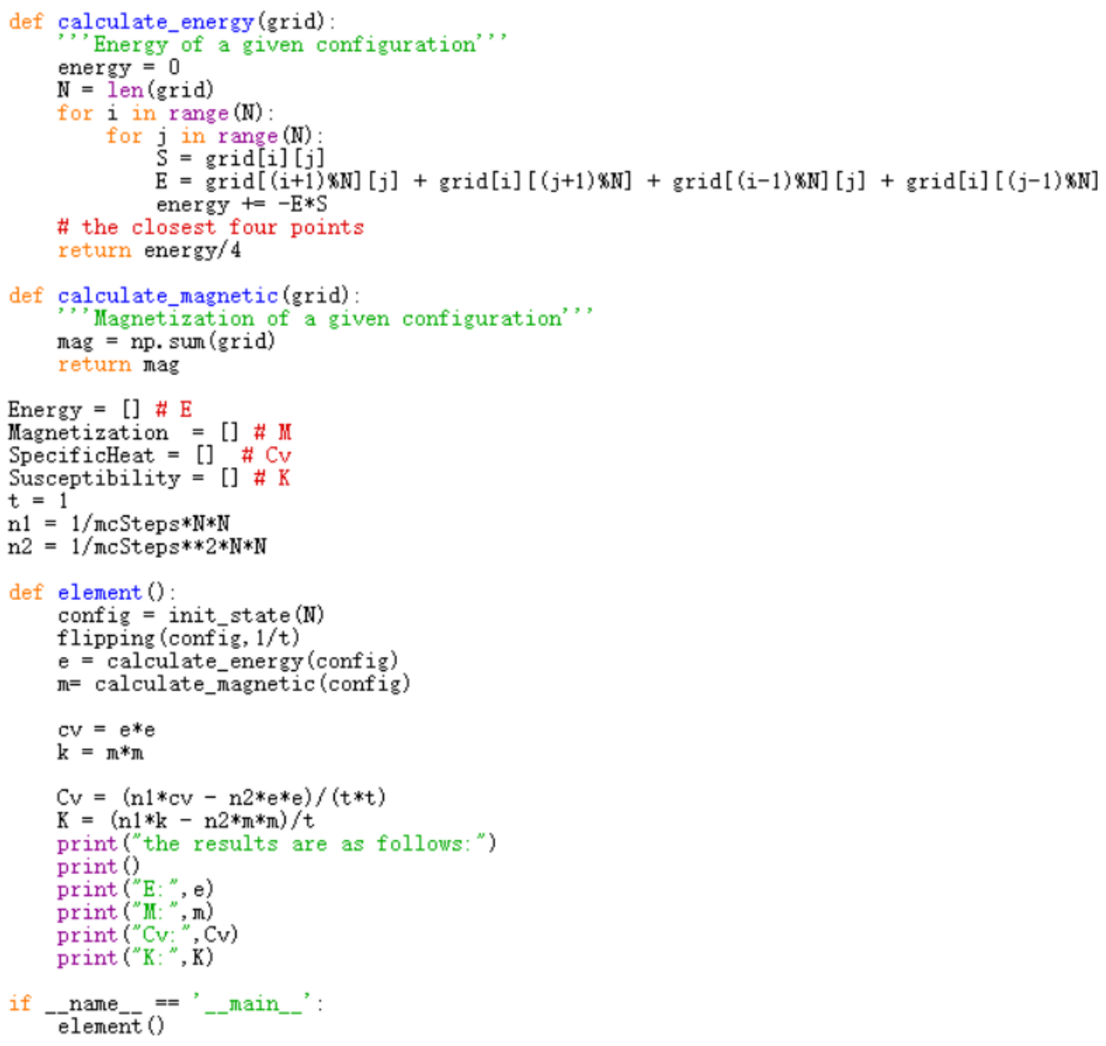
for its specific heat, Cv can represent the partial conductance of the internal energy to the temperature T, and the magnetic susceptibility K, or X, can be roughly expressed as the square of the magnetic moment. The ratio to T. For a system of [[1,1],[-1,1]], it can be calculated that its energy is 0, the magnetic moment is 2, Cv is 0, and the magnetic susceptibility is 0.016.

**4b**

The purpose of the second task is to write a code that can simulate the Ising model and calculate E, M, Cv, K, which is also 2*2 and is compared with the result of a. The structure of the whole code is as shown in the figure below. First, a 2*2 grid is randomly generated, and then a series of operations are performed to obtain the E, M, Cv, and K values of the entire system. Similarly, by adjusting the size of N, we also It is possible to calculate the E, M, Cv, K values under the other lattice L, which is the fifth task.



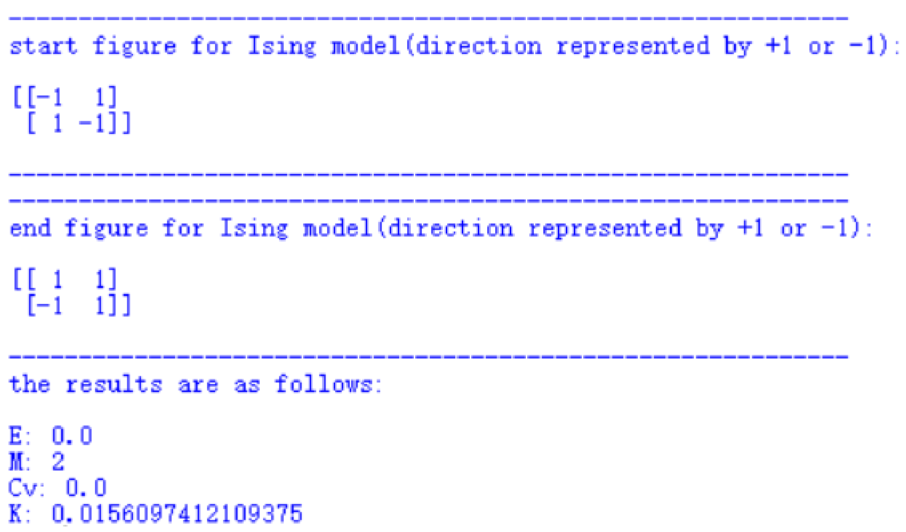
```
import time
import numpy as np
from numpy.random import rand
import matplotlib.pyplot as plt

N       = 2        # lattice number L
eqSteps = 2**10        # equilibrium step
mcSteps = 2**10        # monte carlo step

def init_state(N):
    '''  generates a random spin configuration for initial condition'''
    # np.random.randint(2, size=(N,N)) generate random number and return as a list
    state = 2*np.random.randint(2, size=(N,N))-1
    print("-"*60)
    print("start figure for Ising model(direction represented by +1 or -1):")
    print()
    print(state)
    print()
    print("-"*60)
    return state


def flipping(grid, beta):
    '''Monte Carlo move using Metropolis algorithm '''
    N = len(grid)
    for i in range(N):
        for j in range(N):
            # generate random number from 0 to N-1
            a = np.random.randint(0, N)
            b = np.random.randint(0, N)
            s = grid[a][b]
            E = grid[(a+1)%N][b] + grid[a][(b+1)%N] + grid[(a-1)%N][b] + grid[a][(b-1)%N]
            cost = 2*s*E
            # if the energy fall down, then turn around
            if cost < 0:
                s *= -1
            # generate random number from 0 to 1, if P < exp(-E/(kT)), then turn around
            elif rand() < np.exp(-cost*beta):
                s *= -1
            grid[a][b] = s
    print("-"*60)
    print("end figure for Ising model(direction represented by +1 or -1):")
    print()
    print(grid)
    print()
    print("-"*60)
    return grid
```



```
def calculate_energy(grid):
    '''Energy of a given configuration'''
    energy = 0
    N = len(grid)
    for i in range(N):
        for j in range(N):
            S = grid[i][j]
            E = grid[(i+1)%N][j] + grid[i][(j+1)%N] + grid[(i-1)%N][j] + grid[i][(j-1)%N]
            energy += -E*S
    # the closest four points
    return energy/4

def calculate_magnetic(grid):
    '''Magnetization of a given configuration'''
    mag = np.sum(grid)
    return mag

Energy = [] # E
Magnetization  = [] # M
SpecificHeat = []  # Cv
Susceptibility = [] # K
t = 1
n1 = 1/mcSteps*N*N
n2 = 1/mcSteps**2*N*N

def element():
    config = init_state(N)
    flipping(config,1/t)
    e = calculate_energy(config)
    m= calculate_magnetic(config)

    cv = e*e
    k = m*m

    Cv = (n1*cv - n2*e*e)/(t*t)
    K = (n1*k - n2*m*m)/t
    print("the results are as follows:")
    print()
    print("E:",e)
    print("M:",m)
    print("Cv:",Cv)
    print("K:",K)

if __name__ == '__main__':
    element()
```

Figure 5 Main code for 4b

The running result is shown in the figure below. The calculated E, M, Cv, and K values are the parameters corresponding to the lattice system in the “end figure”, which is [[1,1],[-1,1]] .

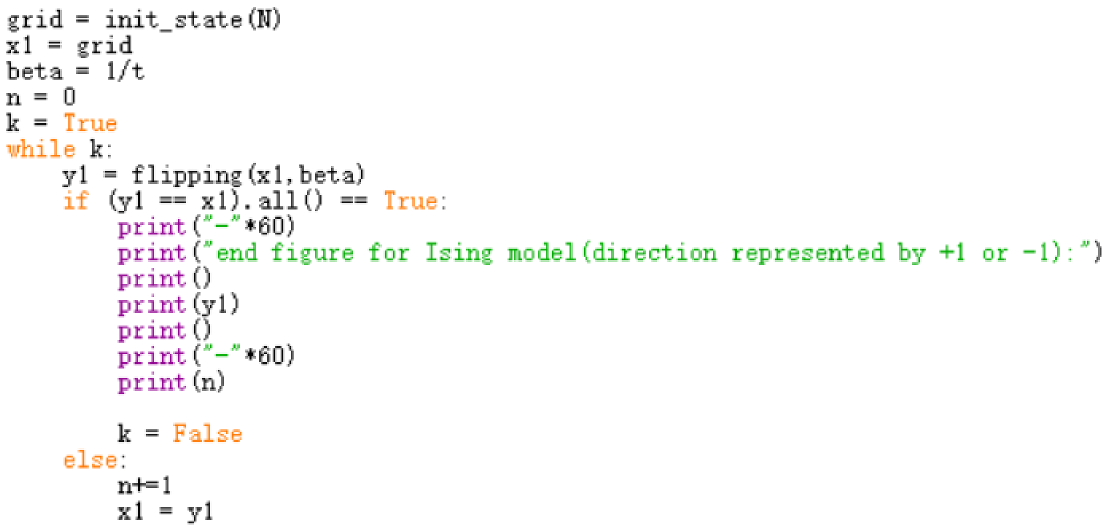


```
-------------------------------------------------------------
start figure for Ising model(direction represented by +1 or -1):

[[-1  1]
 [ 1 -1]]

-------------------------------------------------------------
-------------------------------------------------------------
end figure for Ising model(direction represented by +1 or -1):

[[ 1  1]
 [-1  1]]

-------------------------------------------------------------
the results are as follows:

E: 0.0
M: 2
Cv: 0.0
K: 0.0156097412109375
```

Figure 6 Results for 2*2 lattice

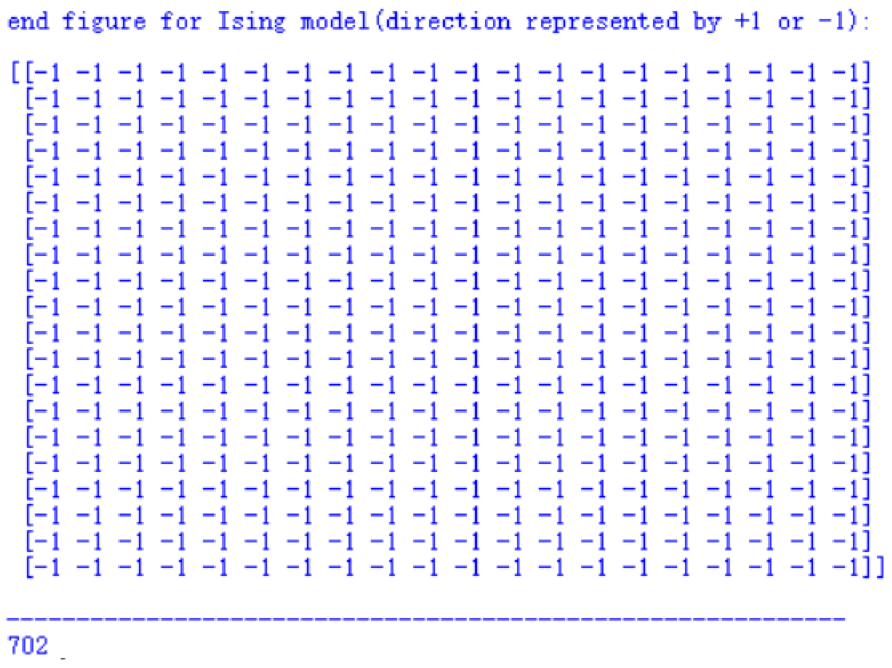
**4c**

The third task is to calculate the time when the 20×20 Ising model reach stable. Calculate in the case of T=1 and T=2.4, draw the relationship between configuration and cycle, and describe its relationship with T. Here, in order to get the time required for the 20*20 lattice Ising model, I added a judgment condition to the code to determine when it reached equilibrium, and also we can find out the monte carlo cycles for reaching equilibrium. The time(or the monte carlo cycles) and the code required to stabilize the Ising model is as shown in the figure. If the Ising model does not change, it indicates that it has stabilized.



```
grid = init_state(N)
x1 = grid
beta = 1/t
n = 0
k = True
while k:
    y1 = flipping(x1,beta)
    if (y1 == x1).all() == True:
        print("-"*60)
        print("end figure for Ising model(direction represented by +1 or -1):")
        print()
        print(y1)
        print()
        print("-"*60)
        print(n)

        k = False
    else:
        n+=1
        x1 = y1
```

Figure 7 Main code for 4c



```
end figure for Ising model(direction represented by +1 or -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 -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 -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 -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 -1]]

------------------------------------------------------------
702
```

Figure 8 Outcome MC cycles for random Ising Model

## 4d

The fourth task is to analyze the probability distribution, that is, the probability distribution of the energy E when the Ising model reaches the equilibrium state, which can be calculated by simply calculating the number of times a given energy appears in the calculation. Once the steady state is reached, the calculation begins. Compare the results with the energy variance and discuss. I simulated the 20*20 grid and obtained the energy of 500 samples in the final steady state, and obtained the energy distribution. The results show that at T=1, the probability of the lowest energy in all samples is the biggest, the other probability is small, as shown in the figure; at T = 2.4, since the phase change has occurred at this time, there is no relatively stable state, and the energy of all samples tends to be normal distributed. As the diagram no.2 in figure 9.

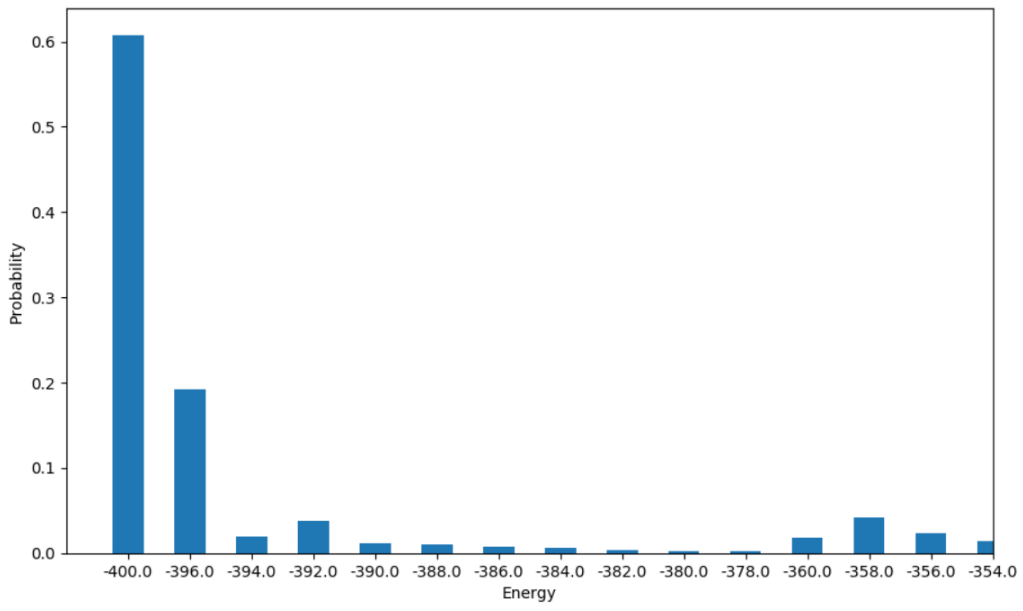


Figure 9 Probabilty distribution for 20*20 lattice at T=1

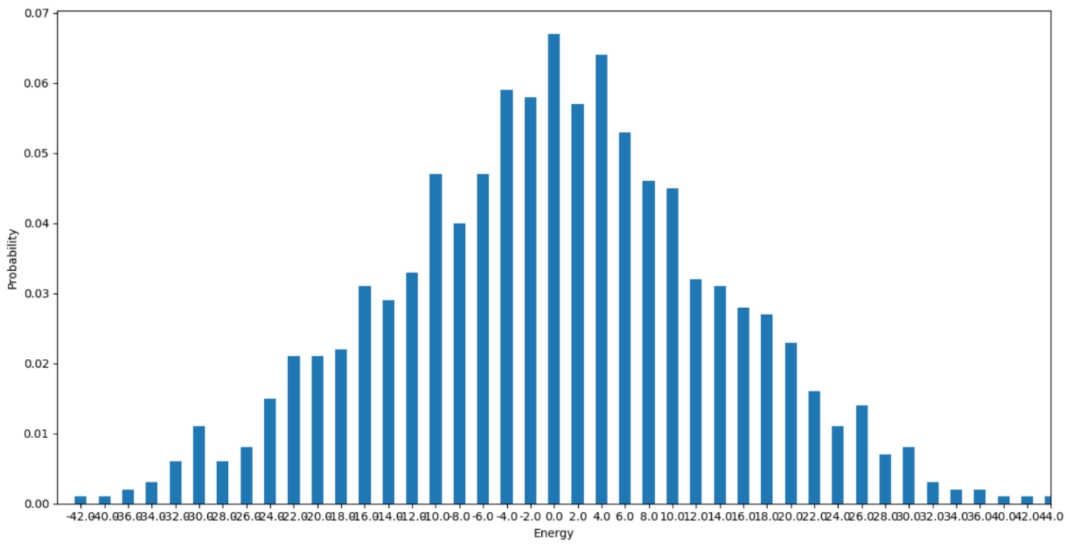
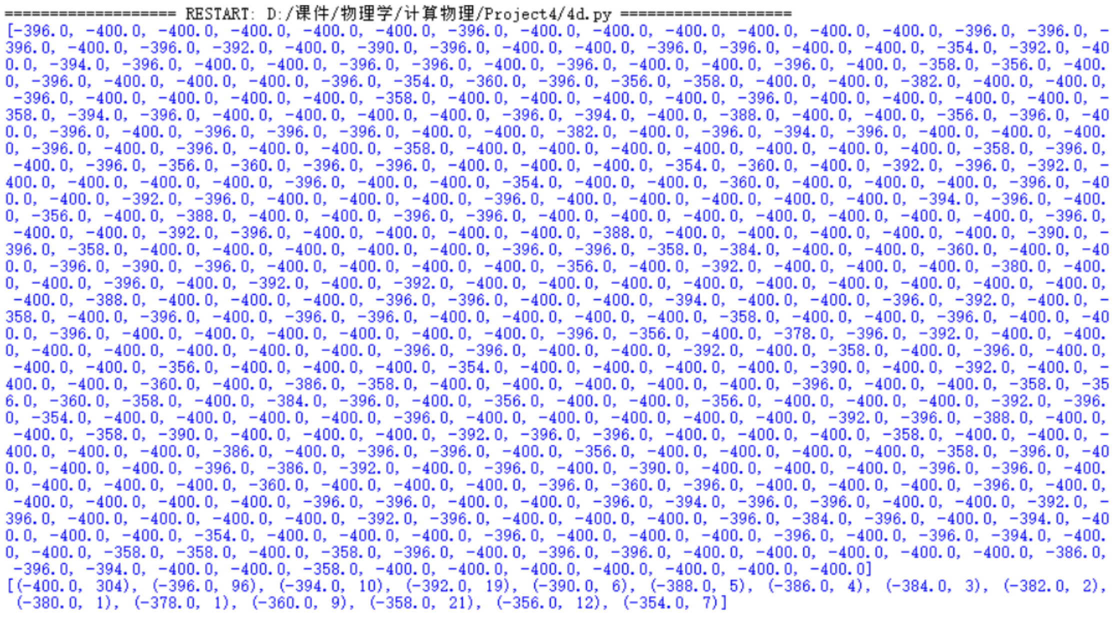


Figure 10 Probability distribution for 20*20 lattice at T=2.4

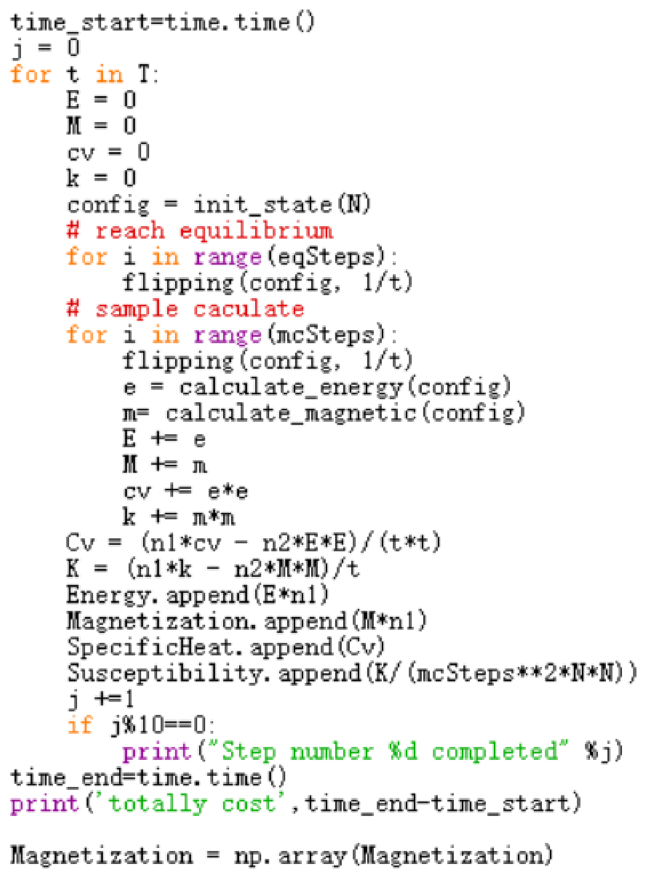


```
=================== RESTART: D:/课件/物理学/计算物理/Project4/4d.py ===================
[-396.0, -400.0, -400.0, -400.0, -400.0, -400.0, -396.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -396.0, -396.0, -
396.0, -400.0, -396.0, -392.0, -400.0, -390.0, -396.0, -400.0, -400.0, -396.0, -396.0, -400.0, -400.0, -354.0, -392.0, -40
0.0, -394.0, -396.0, -400.0, -400.0, -396.0, -396.0, -400.0, -396.0, -400.0, -400.0, -396.0, -400.0, -358.0, -356.0, -400.
0, -396.0, -400.0, -400.0, -400.0, -396.0, -354.0, -360.0, -396.0, -356.0, -358.0, -400.0, -400.0, -382.0, -400.0, -400.0,
 -396.0, -400.0, -400.0, -400.0, -400.0, -358.0, -400.0, -400.0, -400.0, -400.0, -396.0, -400.0, -400.0, -400.0, -400.0, -
358.0, -394.0, -396.0, -400.0, -400.0, -400.0, -400.0, -396.0, -394.0, -400.0, -388.0, -400.0, -400.0, -356.0, -396.0, -40
0.0, -396.0, -400.0, -396.0, -396.0, -396.0, -400.0, -400.0, -382.0, -400.0, -396.0, -394.0, -396.0, -400.0, -400.0, -400.
0, -396.0, -400.0, -396.0, -400.0, -400.0, -358.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -358.0, -396.0,
 -400.0, -396.0, -356.0, -360.0, -396.0, -396.0, -400.0, -400.0, -400.0, -354.0, -360.0, -400.0, -392.0, -396.0, -392.0, -
400.0, -400.0, -400.0, -400.0, -396.0, -400.0, -400.0, -354.0, -400.0, -400.0, -360.0, -400.0, -400.0, -400.0, -396.0, -40
0.0, -400.0, -392.0, -396.0, -400.0, -400.0, -400.0, -396.0, -400.0, -400.0, -400.0, -400.0, -400.0, -394.0, -396.0, -400.
0, -356.0, -400.0, -388.0, -400.0, -400.0, -396.0, -396.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -396.0,
 -400.0, -400.0, -392.0, -396.0, -400.0, -400.0, -400.0, -400.0, -388.0, -400.0, -400.0, -400.0, -400.0, -400.0, -390.0, -
396.0, -358.0, -400.0, -400.0, -400.0, -400.0, -400.0, -396.0, -396.0, -358.0, -384.0, -400.0, -400.0, -360.0, -400.0, -40
0.0, -396.0, -390.0, -396.0, -400.0, -400.0, -400.0, -400.0, -356.0, -400.0, -392.0, -400.0, -400.0, -400.0, -380.0, -400.
0, -400.0, -396.0, -400.0, -392.0, -400.0, -392.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0,
 -400.0, -388.0, -400.0, -400.0, -400.0, -396.0, -396.0, -400.0, -400.0, -394.0, -400.0, -400.0, -396.0, -392.0, -400.0, -
358.0, -400.0, -396.0, -400.0, -396.0, -396.0, -400.0, -400.0, -400.0, -400.0, -358.0, -400.0, -400.0, -396.0, -400.0, -40
0.0, -396.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -396.0, -356.0, -400.0, -378.0, -396.0, -392.0, -400.0, -400.
0, -400.0, -400.0, -400.0, -400.0, -400.0, -396.0, -396.0, -400.0, -400.0, -392.0, -400.0, -358.0, -400.0, -396.0, -400.0,
 -400.0, -400.0, -356.0, -400.0, -400.0, -400.0, -354.0, -400.0, -400.0, -400.0, -400.0, -390.0, -400.0, -392.0, -400.0, -
400.0, -400.0, -360.0, -400.0, -386.0, -358.0, -400.0, -400.0, -400.0, -400.0, -400.0, -396.0, -400.0, -400.0, -358.0, -35
6.0, -360.0, -358.0, -400.0, -384.0, -396.0, -400.0, -356.0, -400.0, -400.0, -356.0, -400.0, -400.0, -400.0, -392.0, -396.
0, -354.0, -400.0, -400.0, -400.0, -400.0, -396.0, -400.0, -400.0, -400.0, -400.0, -400.0, -392.0, -396.0, -388.0, -400.0,
 -400.0, -358.0, -390.0, -400.0, -400.0, -400.0, -392.0, -396.0, -396.0, -400.0, -400.0, -400.0, -358.0, -400.0, -400.0, -
400.0, -400.0, -400.0, -386.0, -400.0, -396.0, -396.0, -400.0, -356.0, -400.0, -400.0, -400.0, -400.0, -358.0, -396.0, -40
0.0, -400.0, -400.0, -396.0, -386.0, -392.0, -400.0, -396.0, -400.0, -390.0, -400.0, -400.0, -400.0, -396.0, -396.0, -400.
0, -400.0, -400.0, -400.0, -360.0, -400.0, -400.0, -400.0, -396.0, -360.0, -396.0, -400.0, -400.0, -400.0, -396.0, -400.0,
 -400.0, -400.0, -400.0, -400.0, -396.0, -396.0, -400.0, -400.0, -396.0, -394.0, -396.0, -396.0, -400.0, -400.0, -392.0, -
396.0, -400.0, -400.0, -400.0, -400.0, -392.0, -396.0, -400.0, -400.0, -400.0, -396.0, -384.0, -396.0, -400.0, -394.0, -40
0.0, -400.0, -400.0, -354.0, -400.0, -400.0, -400.0, -396.0, -400.0, -400.0, -400.0, -396.0, -400.0, -396.0, -394.0, -400.
0, -400.0, -358.0, -358.0, -400.0, -358.0, -396.0, -400.0, -396.0, -396.0, -400.0, -400.0, -400.0, -400.0, -400.0, -386.0,
 -396.0, -394.0, -400.0, -400.0, -358.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0, -400.0]
[(-400.0, 304), (-396.0, 96), (-394.0, 10), (-392.0, 19), (-390.0, 6), (-388.0, 5), (-386.0, 4), (-384.0, 3), (-382.0, 2),
 (-380.0, 1), (-378.0, 1), (-360.0, 9), (-358.0, 21), (-356.0, 12), (-354.0, 7)]
```

Figure 11 Results for 500 samples, the last 2 lines are the number of each energy appearing

## 4e

The fifth task is to simulate different L for Ising model and perform it as a function of T between 2.0 and 2.3. Also calculate E, M, Cv, K for the different grids L at different T. This code is the code introduced in the Method module. We modify the N size to get different grids, and finally get the results. The main code is shown in the figure below.



```
time_start=time.time()
j = 0
for t in T:
    E = 0
    M = 0
    cv = 0
    k = 0
    config = init_state(N)
    # reach equilibrium
    for i in range(eqSteps):
        flipping(config, 1/t)
    # sample caculate
    for i in range(mcSteps):
        flipping(config, 1/t)
        e = calculate_energy(config)
        m= calculate_magnetic(config)
        E += e
        M += m
        cv += e*e
        k += m*m
    Cv = (n1*cv - n2*E*E)/(t*t)
    K = (n1*k - n2*M*M)/t
    Energy.append(E*n1)
    Magnetization.append(M*n1)
    SpecificHeat.append(Cv)
    Susceptibility.append(K/(mcSteps**2*N*N))
    j +=1
    if j%10==0:
        print("Step number %d completed" %j)
time_end=time.time()
print('totally cost',time_end-time_start)

Magnetization = np.array(Magnetization)
```

Figure 12 Main code for 4e

### 4f

By obtaining Tc under different lattices, we can fit these data using linefit, because with v=1, the final fitting result can get the Curie temperature Tc at when L approaches infinity.

## 4. Results

### 4a

We obtained the analytical solution of the 2*2 lattice Ising model and get the E, M, Cv, K. The formula is as follows. For a system of [[1,1],[-1,1]], it can be calculated that its energy is 0, the magnetic moment is 2, Cv is 0, and the magnetic susceptibility is 0.016.

$$E = -J(s_{11}s_{12} + s_{11}s_{21} + s_{22}s_{12} + s_{22}s_{21})$$

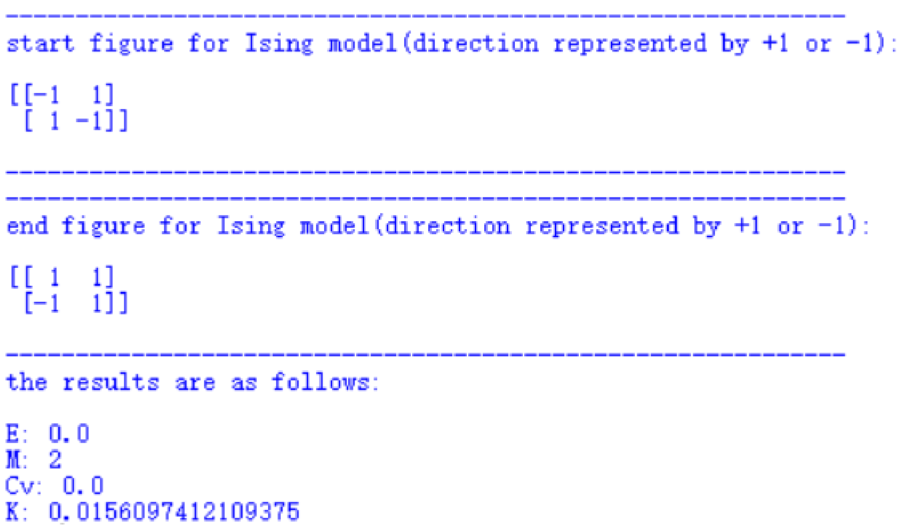
$$\mathrm{M} = \mathrm{s}_{11} + s_{12} + s_{21} + s_{22}$$

$$\mathrm{Cv} = \partial\mathrm{E}/\partial\mathrm{T}$$

$$\mathrm{K} = \mathrm{M}/\mathrm{H}$$

### 4b

We obtained the code that can simulate the Ising model under different grids, and obtained the values of E, M, Cv, K by taking the 2*2 grid as an example. The code and result are as follows. By the comparison, the values E, M, Cv, K are the same as in 4a and are well validated.

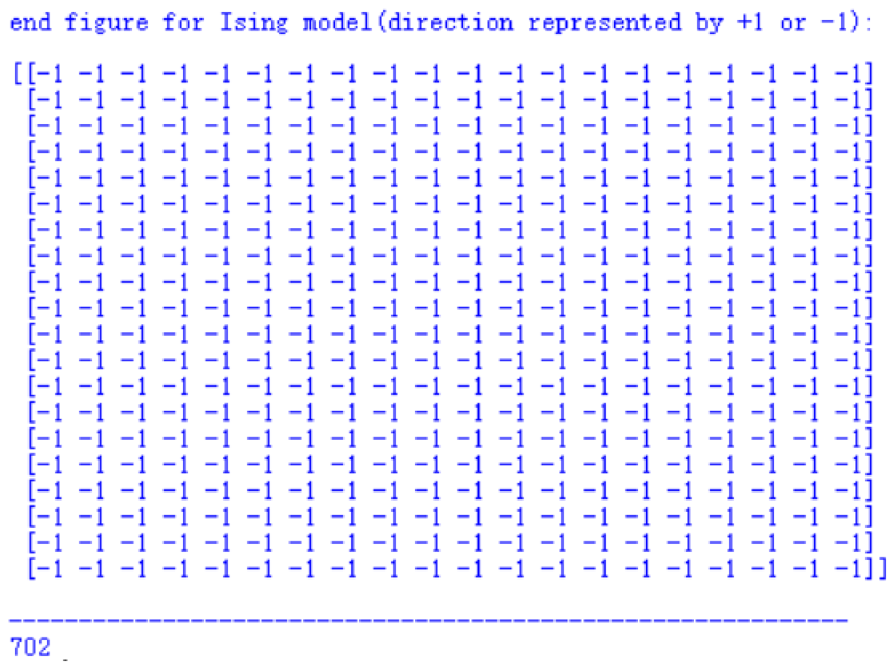


```
------------------------------------------------------------
start figure for Ising model(direction represented by +1 or -1):

[[-1  1]
 [ 1 -1]]

------------------------------------------------------------
------------------------------------------------------------
end figure for Ising model(direction represented by +1 or -1):

[[ 1  1]
 [-1  1]]

------------------------------------------------------------
the results are as follows:

E:  0.0
M:  2
Cv:  0.0
K:  0.0156097412109375
```

Figure 13 Results for 4b

**4c**

At T = 1, we obtained the number of monte carlo cycles required to steady state and obtained the situation when it was stable. At the same time, the number of corresponding cycles for each configuration is also drawn when you run the code, Just like the figure below, the fist diagram is the configuration and the last number is the monte carlo cycles. When T=2.4, since the phase transition temperature is exceeded, there is no phase change, no net magnetic moment is generated, and the spin orientation is very confusing. Therefore, the steady state is not reached, and an image of the number of cycles corresponding to different configurations cannot be drawn.



```
end figure for Ising model(direction represented by +1 or -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 -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 -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 -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 -1]]

------------------------------------------------------------
702
```

Figure 14 Results for 4c

### 4d

I simulated the 20*20 grid. At T=1, we obtained the energy of 500 samples in the final steady state, and obtained the energy distribution P(E). It was found that in most cases, the final steady state was where the energy tends to the lowest value, and the probability of other cases is small relative to it. At T=2.4, since it exceeds the Curie temperature, the spin orientation is disordered, there is no final stable state, and the energy distribution P(E) is theoretically normal distributed like the figure 15.

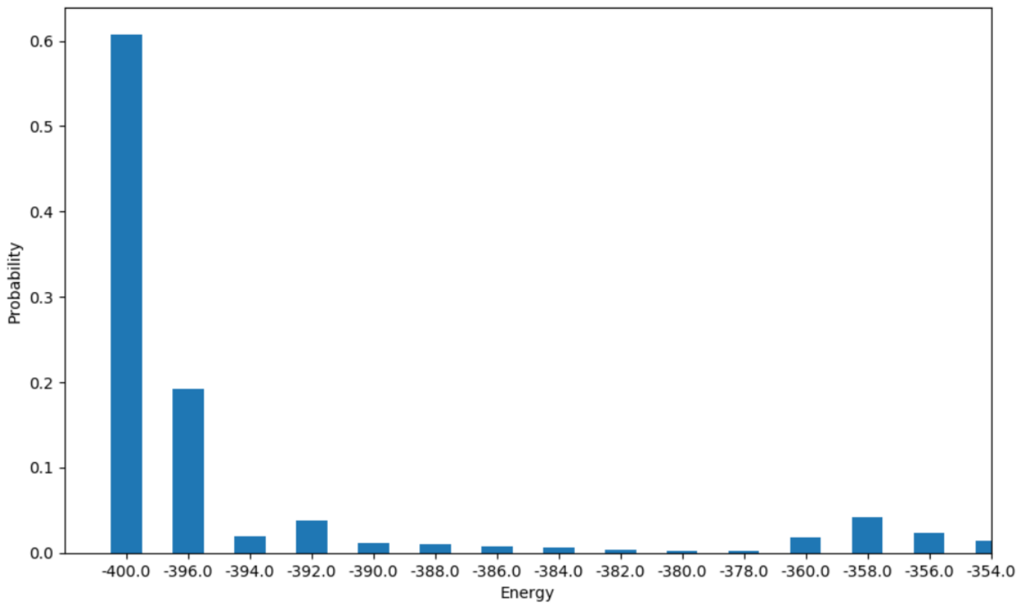


Figure 15 Results for 4d at T =1

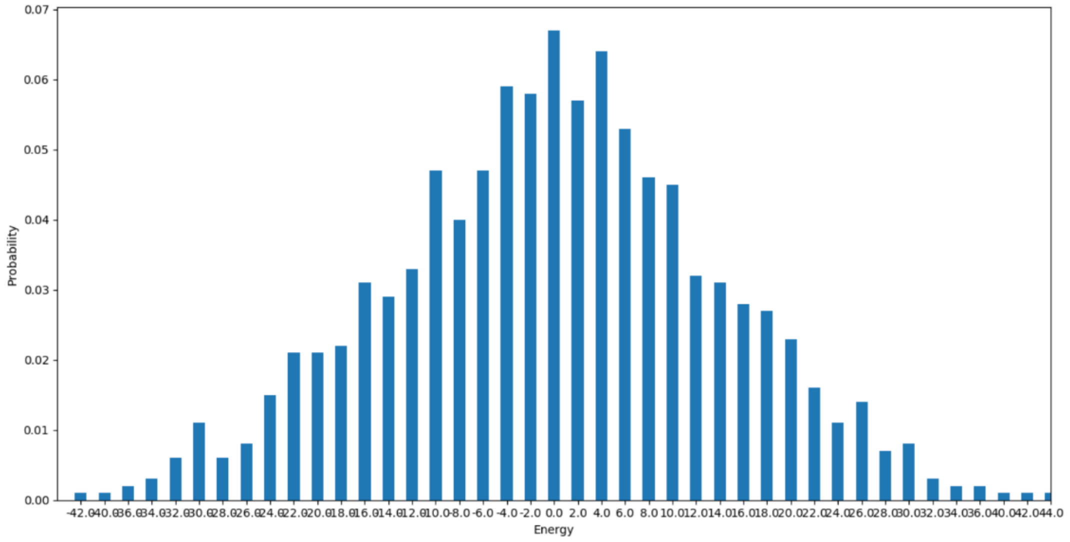


Figure 16 Results for 4d at T =2.4

## 4e

We obtained the values of E, M, Cv, and K under different L by simulating the Ising model under different T. The results are shown in the figure. By segmentally sampling T in the interval of 2.0-2.3, the relationship between the values of E, M, Cv, K and T is obtained and the figures can be drawn.

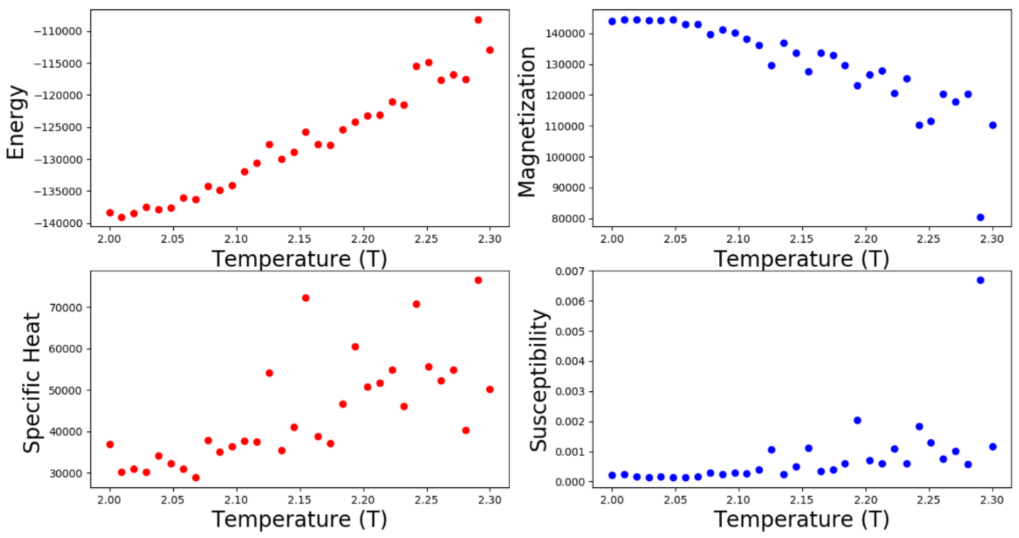


Figure 17 L=20, elements change with the function of T

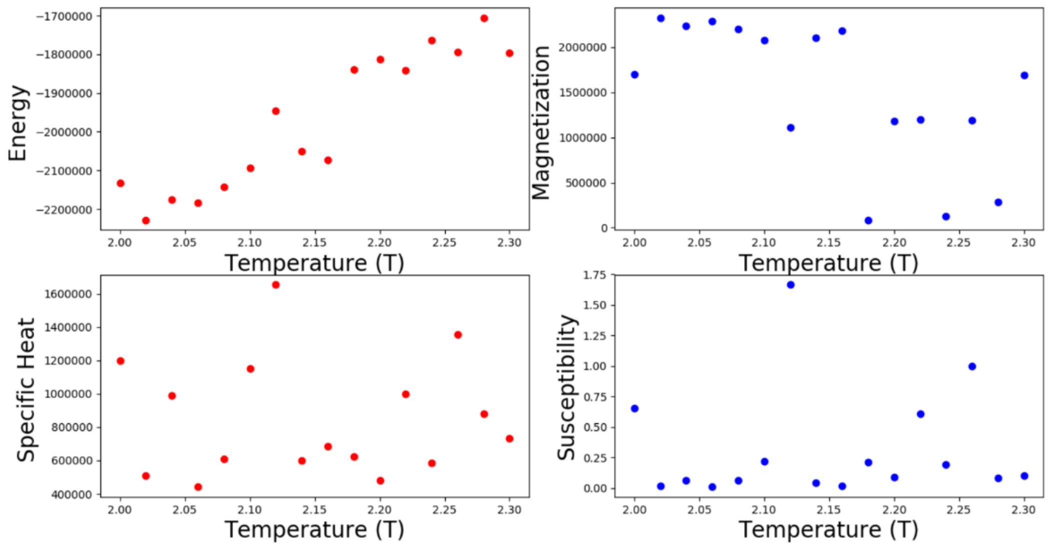


Figure 18 L=40, elements change with the function of T

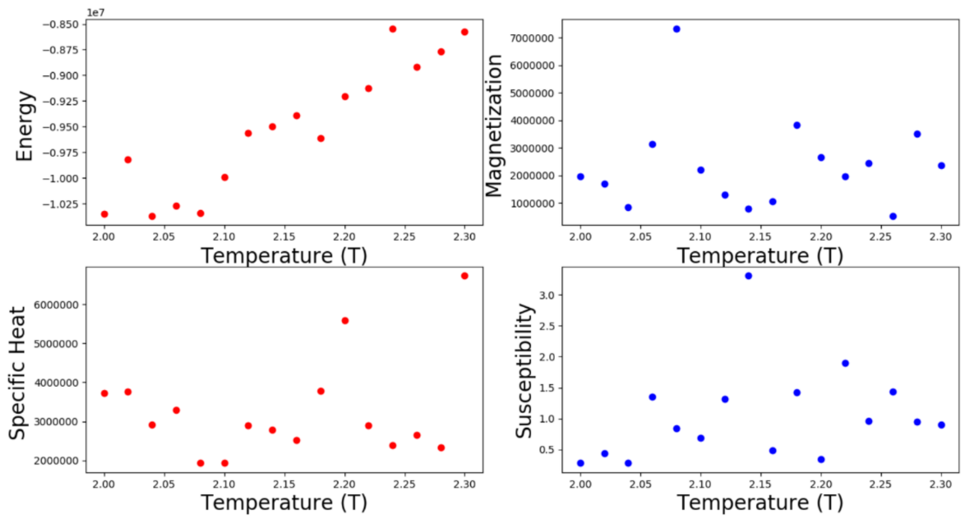


Figure 19 L=60, elements change with the function of T

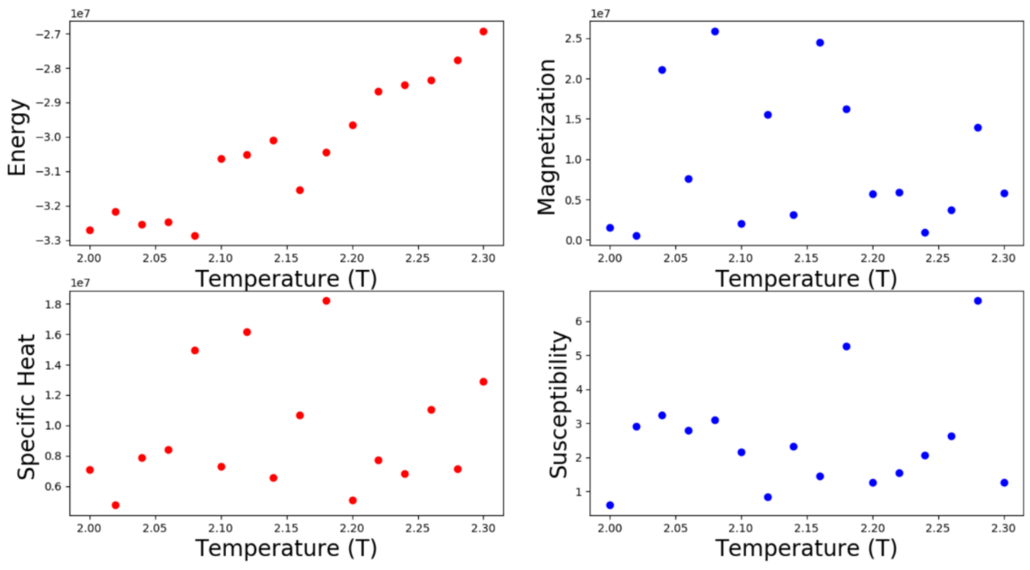


Figure 20 L=80, elements change with the function of T

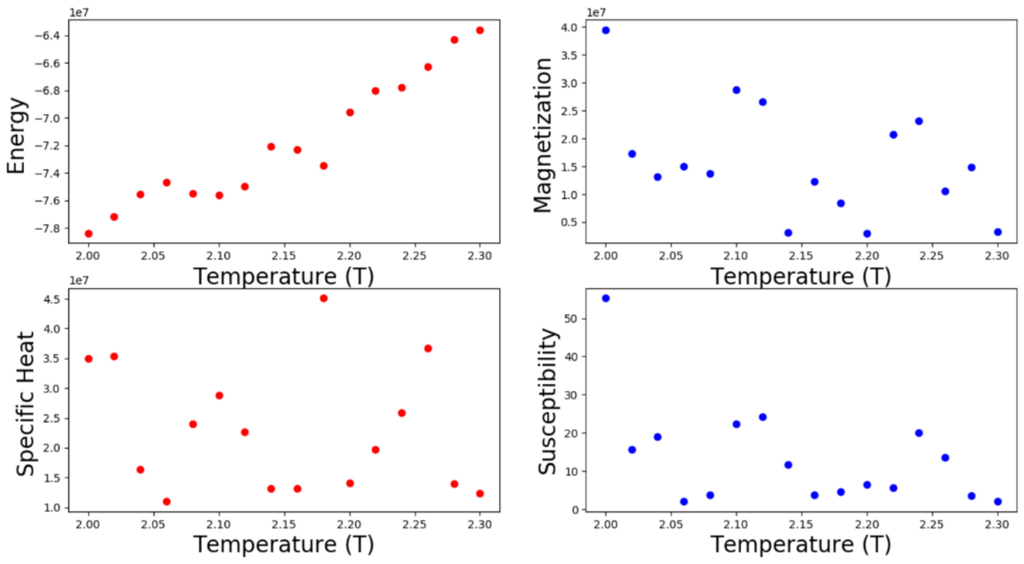
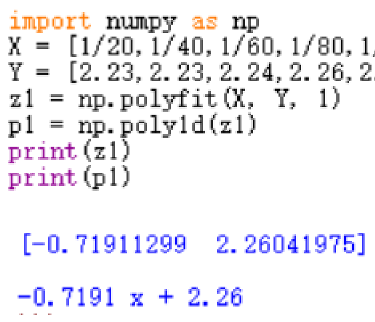


Figure 21 L=100, elements change with the function of T

### 4f

From the result of 4e, we can get the Tc under different L, so that the Tc when L approaches infinity can be obtained by the fitting those data. The data points and the fitted image are shown in the figure, and finally the Tc is obtained, and by comparing with the theoretical value, the result is nearly similar to 2.269.

The data list is like this [(20, 2.23), (40, 2.23), (60, 2.24), (80, 2.26), (100, 2.26)]. In the end, we get the result that Tc is approximately 2.26.



```
import numpy as np
X = [1/20,1/40,1/60,1/80,1/100]
Y = [2.23,2.23,2.24,2.26,2.26]
z1 = np.polyfit(X, Y, 1)
p1 = np.poly1d(z1)
print(z1)
print(p1)
```

```
[-0.71911299  2.26041975]

-0.7191 x + 2.26
```

Figure 22 Results for Tc approaching infinity

## 5. Concluding Remarks

This project has done a good job of simulating the Ising model in different situations and successfully calculated the values of E, M, Cv, K under each Ising model.

The results were good, and the simulated images represented by the matrix were successfully obtained and the tasks required for the project were completed. Meanwhile data was also obtained. However, due to the limitations of Python itself, data overflow sometimes occurs when calculating 4e. Therefore, the program can only continue to run by reducing the number of operations, thereby limiting the accuracy of the result, but because the number of samples is still enough, so the impact will not be too great. Due to program limitations, sometimes the code of each part will run slow. In general, the program has achieved various functions, completed various task requirements, and achieved good results.

## 6. Reference

Hjort-Jensen,M., 2015. Computational physics.

The program can be found in https://github.com/lch172061365/Computational-Physics.git