Statistical Mechanics Individual Project - 2D Ising Model Simulation

Weijun Yuan

(Dated: December 8, 2021)

Abstract

This project discusses the simulation of the Ising model using Monte-Carlo method and studies its properties using the methods of finite size scaling.

I. PROBLEM 1-METROPOLIS ALGORITHM

The condition of detailed balance is summarized by the following expression (Equation 7.7 from the book):

$$P_\sigma W(\sigma \to v) = P_v W(v \to \sigma). \tag{1}$$

If the configuration obeys the Boltzmann distribution, the condition of the detailed balance is given by

$$\frac{W(v \to \sigma)}{W(\sigma \to v)} = \frac{P_\sigma}{P_v} = e^{-\beta(E_\sigma - E_v)}. \tag{2}$$

One way to fulfill the detailed balance is the Metropolis algorithm. The Metropolis algorithm reads:

$$W(v \to \sigma) = 1 \quad \text{for} \quad P_\sigma \geq P_v \quad (E_\sigma \leq E_v), \tag{3}$$

$$W(v \to \sigma) = \frac{P_\sigma}{P_v} \quad \text{for} \quad P_\sigma < P_v \quad (E_\sigma > E_v). \tag{4}$$

To show that Metropolis algorithm fullfil the detailed balance, firstly we assume that for initial and final configuration i and f, $P_f > P_i$. Then, according to Metropolis, $W(i \to f) = 1$ and $W(f \to i) = P_i/P_f$. Therefore,

$$W(f \to i)\frac{P_f}{P_i} = 1 = W(i \to f). \tag{5}$$

Then we have

$$P_f W(f \to i) = P_i W(i \to f), \tag{6}$$

which is the same as the condition of detailed balance (Equation 1). If we assume that $P_f < P_i$, then Equation 3 and 4 give $W(i \to f) = P_f/P_i$ and $W(f \to i) = 1$, which suggests that

$$W(i \to f)\frac{P_i}{P_f} = 1 = W(f \to i). \tag{7}$$

Then we have

$$P_i W(i \to f) = P_f W(f \to i). \tag{8}$$

This is again the condition of detailed balance, then we can conclude that Metropolis algorithm fulfill the detailed balance.

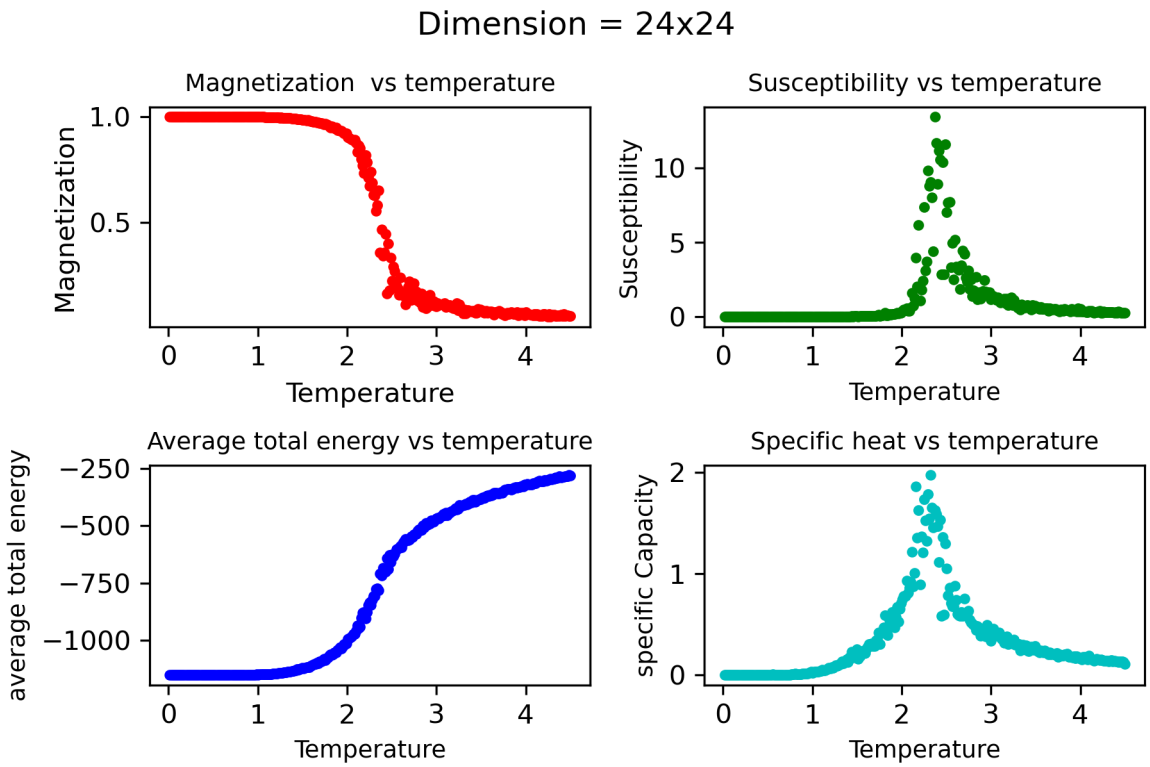

FIG. 1: The absolute magnetization M is defined as $< |\sum S_i| > /N$, the average energy E, the susceptibility and the heat capacity of 24x24 is summarized.

II. PROBLEM 2 - 2D ISING MODEL

I use python to build the 2D Ising model and the code is attached to the report. Since this project is not studying the effect of magnetic field, I only consider B = 0. 1x10^5 sweeps are conducted to allow the system to reach equilibrium and 3x10^4 measurements for each 10 sweeps for each temperature are done to make sure a reasonable statistical result. The measurements of this simulation average the average energy, the specific heat, magnetization and the magnetic susceptibility. I calculated those observables at different temperatures and Fig. 1 shows the result of 24 x 24 lattice. A detailed discussion about the dimensions is given in the sequential problems.

III. PROBLEM 3 - AVERAGE ENERGY

The plot of average total energy and the average energy per site are shown in Fig 2 and 3 respectively. From the plots, we can see that at low temperature, the average energy per

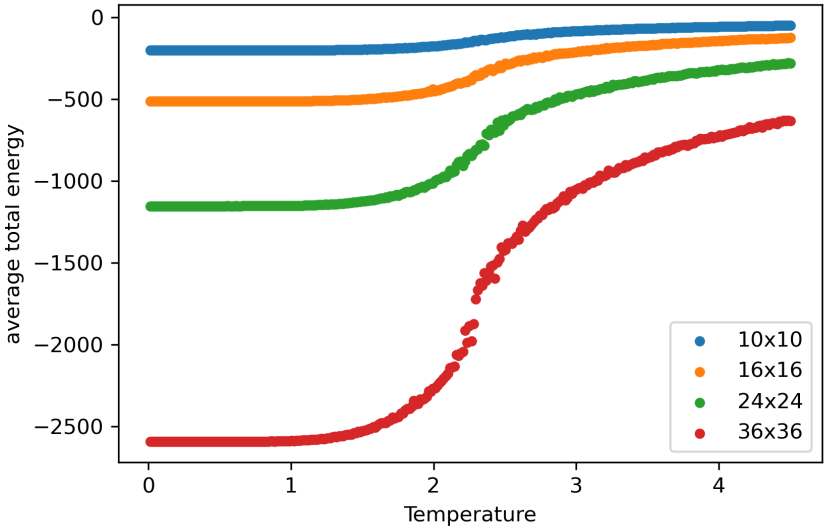

FIG. 2: average total energy versus temperature for different system size

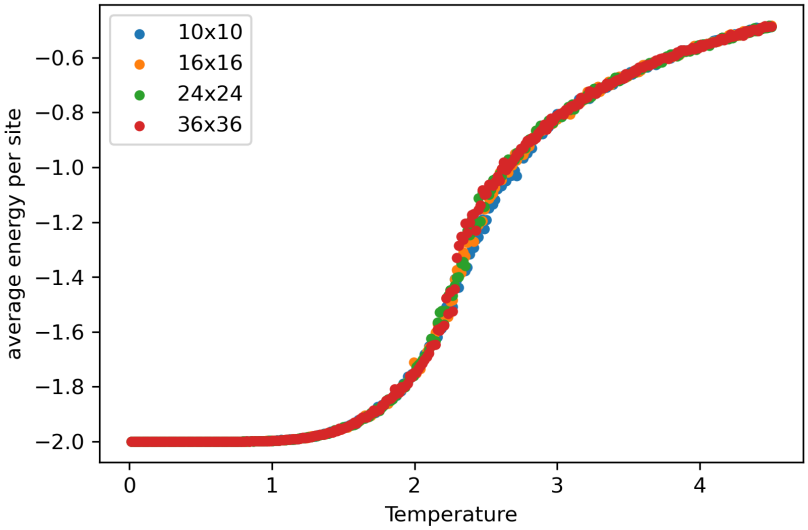

FIG. 3: average energy per site

site converges to -2. This is because that the spin align with each other at low temperature to reduce the energy (ferromagnetism). In $2 \leq T \leq 3$, the energy of the system increases abruptly with the temperature. There is an infection point between $T = 2$ and $T = 3$. This suggests that there can be a phase transition between $2 \leq T \leq 3$. Also, it is noted that the infection point shifts to the left when the dimension increases.

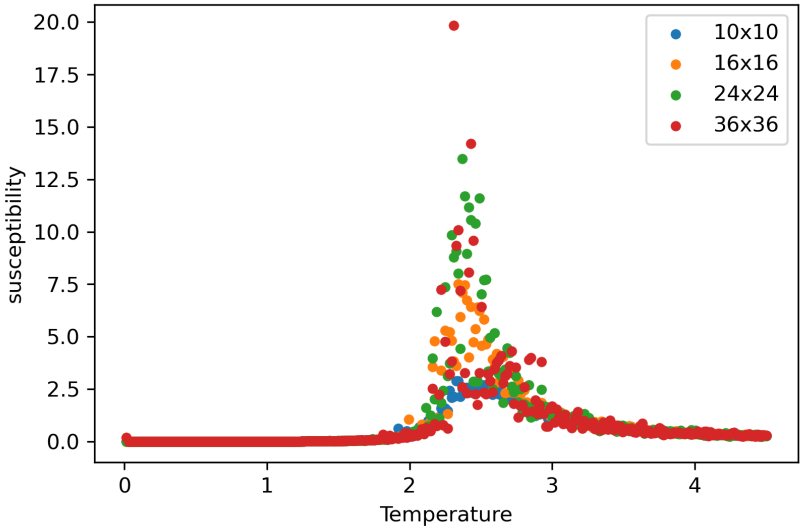

FIG. 4: The susceptibility as a function of T for the four systems. There is a peak between $2 \leq \mathrm{T} \leq 3$ for different systems. In general, the peak value is higher for larger system.

IV. PROBLEM 4 - SUSCEPTIBILITY

The plot of susceptibility as a function of T for the four systems is shown in Fig 4. We noted that the height and position of the peaks change with system size. Table I summarize the peak values as well as Tc(L) for different systems.

TABLE I: The position and height of the peaks in susceptibility for different system sizes.

Dimension(LxL)	Peak position $T_c(L)$	Peak height
10x10	2.475	2.90
16x16	2.34	7.51
24x24	2.37	13.47
36x36	2.31	19.8

Equation 7.88 in the book suggests the following relation between T_c and $T_c(L)$ reads

$$T_c(L) = T_c + (x_0 T_c) L^{-1/v} \tag{9}$$

To verify this relation, we plot $T_c(L)$ vs L^{-1}, where v = 1 is assumed. The plot is shown in Fig 5. The plot roughly follows a linear relationship, which means that our simulation agree

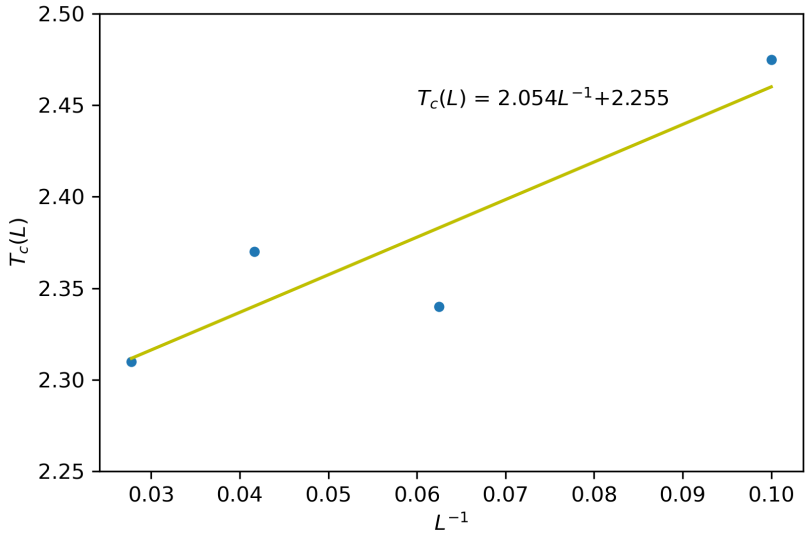

FIG. 5: $T_c(L)$ vs L^{-1}. It looks like a straight line, which means our result agrees quite well with Equation 7.88 if we assuem v = 1.

.

with Equation 7.88 in the book if we assume v = 1. We obtain $Tc(L) = 2.054L^{-1} + 2.255$ from the linear fit, which is shown in the plot with the yellow line. This gave us $T_c = 2.255$, which is very close to $T_c = 2.269185....$The percentage difference is only 0.63%. I also try to do the fitting to Equation 9 without assuming the value of v and x_0. This gives us T_c = 2.233 and v = 0.133. Since we only have 4 data points, this method is not reliable. We need results from larger systems to do a better analysis with this method.

V. PROBLEM 5 - SUSCEPTIBILITY VS DIMENSION

Equation 7.90 in the book suggests that the susceptibility at $T_c(L)$ follows

$$\chi(T = T_c(L)) \propto L^{\gamma/v}. \tag{10}$$

We expect a power law if we plot $\chi(T_c(L))$ versus L on log-log scale.The plot is shown in Fig .6. The power law is obvious as we see that the plot follows a linear relation. If we do a linear fit, we can get a fitted straight line shown in the plot. The slope gives $\gamma/v = 1.505$, which is smaller than the exact value $\gamma/v = 1.75$. The deviation is around 14 %.

The next part of this problem is to plot $L^{-\gamma/v}\chi$ as a function of the scaling variable

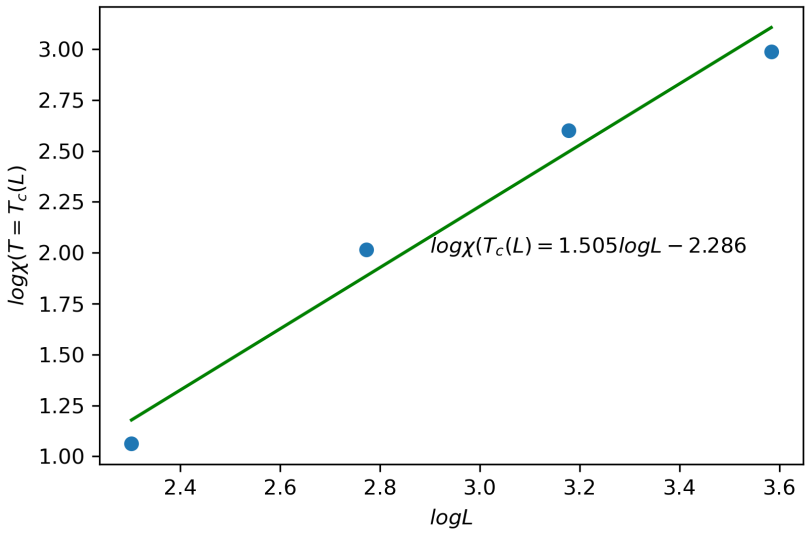

FIG. 6: log $\chi(T = T_c(L)$ vs log L. We expect there should be a power law, so we see a linear plot with the slot $= \gamma/v$. The blue dots shows the data points and the green curve is the result of linear fit.

$L^{1/v}(T - T_c(L))$. According to the book, the results from different system sizes should collapse on the same curve in the critical region if the exponents used are the rights ones. I use the exact value of v = 1 fand $\gamma/v = 1.505$ for the plot and it is shown in Fig. 7. The results do collapse to the same curve in the critical region as the curves for different sizes have the same width and similar peak.

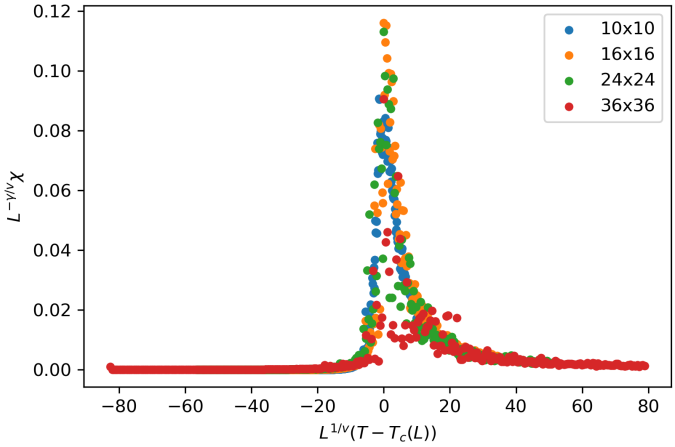

FIG. 7: $L^{-\gamma/v}\chi$ as a function of the scaling variable $L^{1/v}(T - T_c(L))$. $\gamma/v = 1.504$ and v = 1 are used to generated this plot

VI. PROBLEM 6 - MAGNETIZATION

The plot of magnetization versus T for different dimensions is shown in Fig. 8. We observe that the critical temperature $T_c(L)$ decreases when the system size increases. Then, to verify Equation 7.90 in the book, we plot $L^{\beta/v}M$ as a function of T for different size simulated. Fig 9 shows the result if the exact value $\beta/v = 0.25$ is used. I tried different values of β/v and found out the exact value gives a reasonably good intersection at one point of all the magnetization curves. The intersection point locates at T = 2.19, which is slightly lower than the exact value of T_c with 3.5 % deviation.

The next part of this question is to plot $L^{\beta/v}M$ versus the scaling variable $L^{1/v}(T - T_c')$, $T_c' = 2.19$ obtained from previous plot. The values of v = 1 and $\beta/v = 0.25$ are used to generate the plot. The result is presented in Fig. 10 As expected, we see that the magnetization curves collapse onto a single curve in the critical region near T_c(the region slightly above 0).

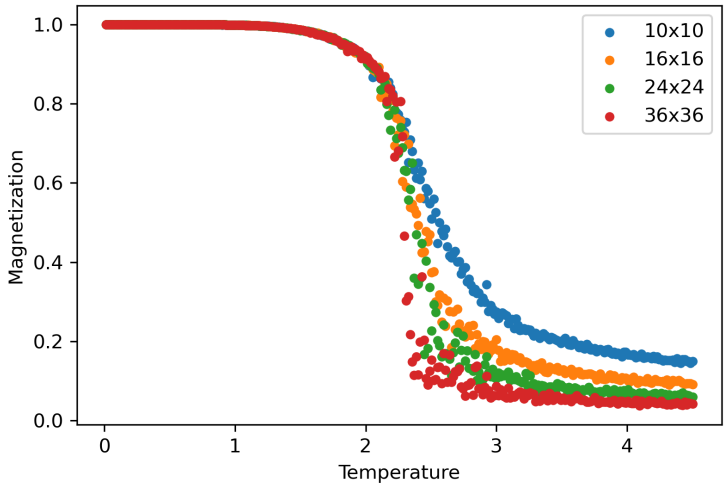

FIG. 8: The magnetization versus temperature for different system sizes.

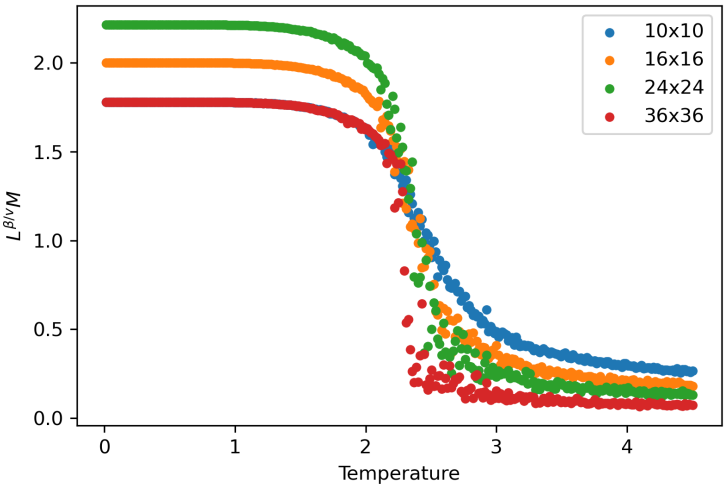

FIG. 9: $L^{\beta/\upsilon}M$ as a function of T for different size simulated. The value $\beta/\upsilon = 0.25$ is used and the curves intersect at T = 2.19

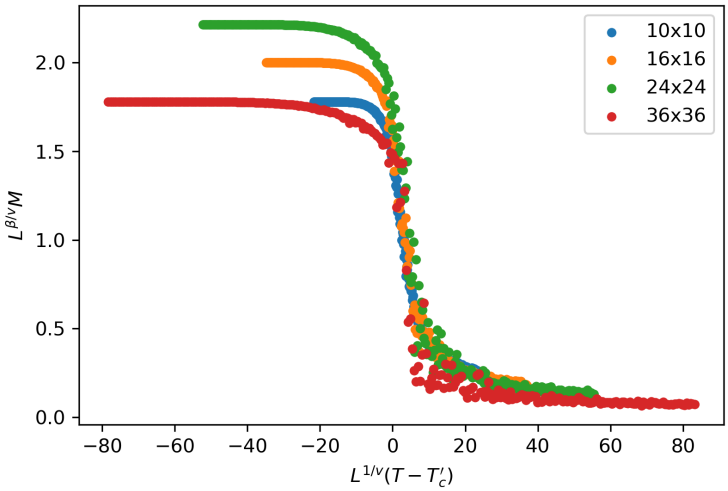

FIG. 10: $L^{\beta/v}M$ as a function of the scaling variable $L^{1/v}(T - T_c')$ for different size simulated. $T_c' = 2.19$, $\beta/v = 0.25$ and v = 1 to generate this plot

VII. PROBLEM 7 - SPECIFIC HEAT

The plot C versus T for the systems simulated is shown in Fig. 11. There is a peak for each system. The position of the peaks $T_c'(L)$ and the height of the peaks C_{max} are shown in Table II.

TABLE II: The position and height of the peaks in specific heat C for different system sizes.

Dimension(LxL)	Peak position $T_c'(L)$	C_{max}
10x10	2.325	1.40
16x16	2.34	1.70
24x24	2.325	1.97
36x36	2.22	1.84

Fig. 12 presents the plot of $T_c'(L)$ versus $L^{-1/v}$, where v is again taken to be 1 again. To find the thermodynamic critical temperature T_c, $T_c'(L)$ and $L^{-1/v}$ are fitted to a linear relation and the y-intersect is the fitted value of T_c. The linear curve is presented with the plot for reference. The fitted value of T_c is 2.239. Compared with the exact value, it is smaller and the deviation is 1.32%, which is larger than the case in magnetic susceptibility. Compared with the value obtained from magnetic susceptibility,it is also smaller and their

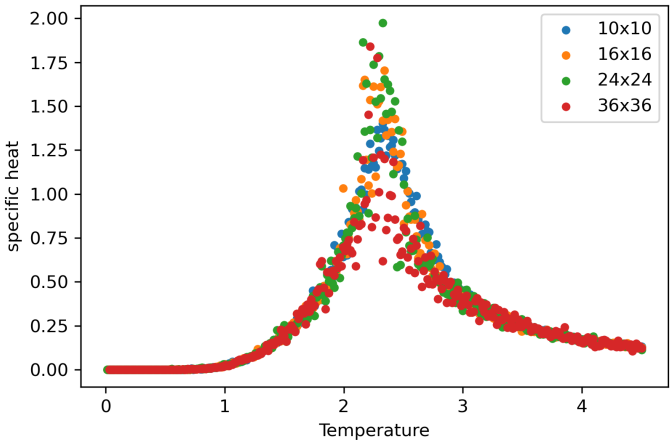

FIG. 11: the specific heat C versus T

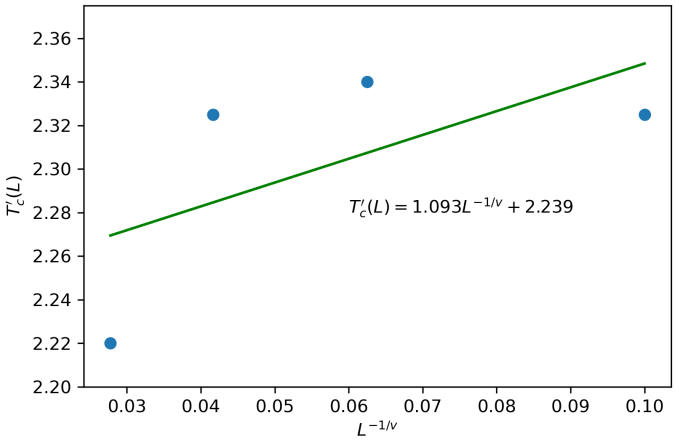

FIG. 12: $T_c'(L)$ versus $L^{-1/v}$. Here, v = 1. A linear fit used to find T_c is shown for reference.

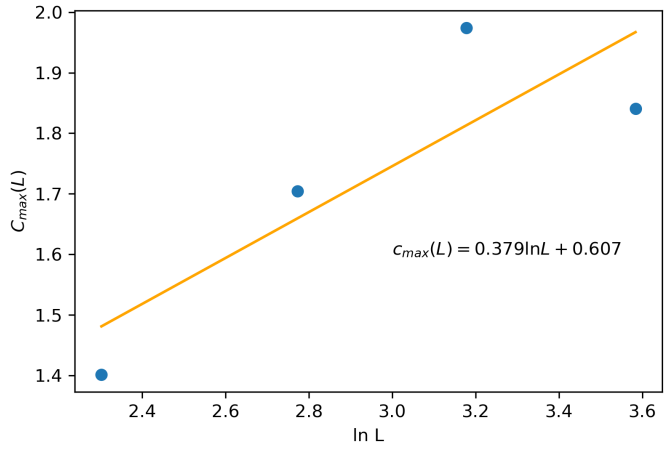

FIG. 13: $C_{max}(L)$ versus $\ln L$. A linear curve fitted to the data is shown for reference

difference is 0.49 %. Therefore, these three values agree very well with each other.

The divergence of C is expected to be logarithmic. To verify this, the plot of $C_{max}(L)$ versus $\ln L$ is shown in Fig.13. A linear curve fitted to the data is also shown in the plot. The result shows that $C_{max}(L)$ and $\ln L$ have a linear relation to some extent. Judging from these 4 data points, we cannot firmly conclude that the divergence of C is logarithmic because the plot do not follow a linear relation completely. More data points are required to make a stronger conclusion.

VIII. PROBLEM 8 -ENTROPY

The entropy can be calculated by the following integral:

$$S(T) = \int_0^T dT' \frac{C(T')}{T'}. \tag{11}$$

I do the integration numerically from T = 0.015 to T =4.5 using trapezoidal rule for different system sizes. For the Ising model, S(T$\rightarrow \infty$) = $\ln 2$ = 0.69314718055.... The results for different dimensions are summarized in Table III. For small system sizes, the results agree quite well with the exact value (deviation $< 10\%$). Since we only integrate up to T = 4.5, the deviation is expected. However, the entropy per spin for 36x36 system has about 20 % from the exact value. The plot of $S(T)$ vs T is shown in Fig. 14. It is clear that the S(T) for 10x10,16x16,24x24 is basically the same. However, the result of 36x36 does not align with other system sizes. I think the reason of it is that 36x36 system may not be completely thermalized after 10^5 sweeps.

TABLE III: The entropy for different system sizes obtained from numerical integrating Equation 11.

Dimension(LxL)	Entropy	deviation from $S(T \rightarrow \infty)$
10x10	0.626	9.6 %
16x16	0.628	9,3 %
24x24	0.628	9.3 %
36x36	0.557	19.6 %

In Fig. 15, the free energy per spin $F = E - TS(T)$ of different systems is present. For all the system sizes, there is a minimum around T = 1.9 and a maximum around T = 2.6. Between these two points, there is a infection point, which should correspond to the critical temperature. The infection point shift to the left when the dimension increases.

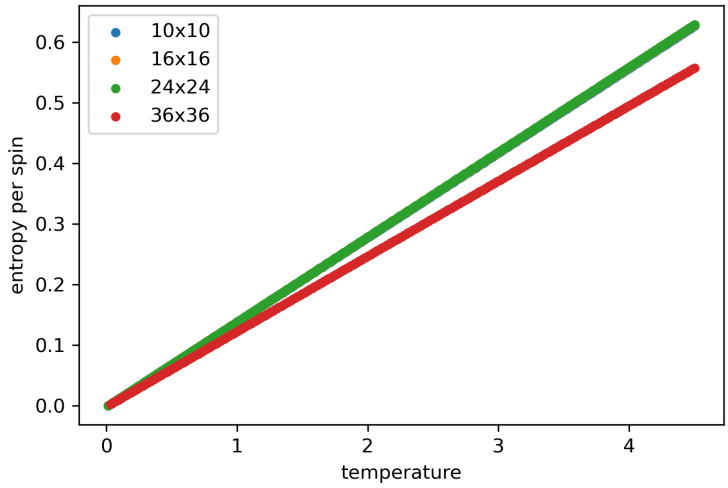

FIG. 14: Entropy per spin for different system sizes. For 10x10,16x16 and 24x24, they has the same S(T). However, 36x36 does not align with other systems, especially at high temperature.

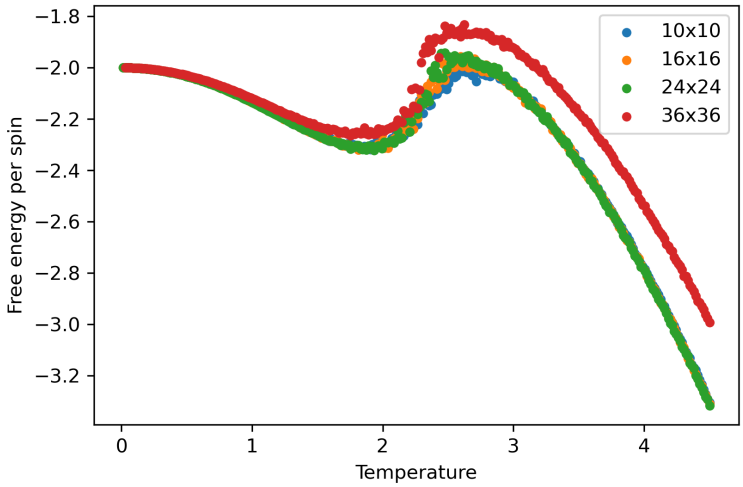

FIG. 15: Free energy per spin of different systems size.

IX. CODE

A. Main program

```
import random
from pylab import plot,figure,errorbar,hist,subplot,title,xlabel,ylabel,legend
```

```
import matplotlib.pyplot as plt
from numpy import mod,zeros,mean,arange,sign, exp,save,array
import time
## ising model


J = 1
L_x = L_y = 10 #dimension
low_T = 0.015*J      #lower temperature limit
high_T = 4.5*J    #higher temp limit
delta_T=0.015*J


latt = zeros([L_x,L_y])
# initialize the spin configuration
for num_x in range(L_y):
    for num_y in range(L_y):
        latt[num_x,num_y] = random.choice([-1,1])
        latt[num_x,num_y] = sign(random.random()-0.5)
# calculate the energy difference
def get_delta_E(nx,ny):
    nx_ml= mod(nx-1,L_x)   # to fulfill the periodic boundary condition
    nx_pl = mod(nx+1,L_x)
    ny_ml = mod(ny-1,L_y)
    ny_pl = mod(ny+1,L_y)
    delta_E = 2*J*latt[nx,ny]*(latt[nx_ml,ny]+\
    latt[nx_pl,ny]+latt[nx,ny_ml]+latt[nx,ny_pl])

    return delta_E

# Update the spin configuration using local update
def one_sweep(T):
```

```
    for num_x in range(L_x):
        for num_y in range(L_y):
            if exp(-get_delta_E(num_x,num_y)/T) > random.random():
                latt[num_x,num_y] = -latt[num_x,num_y]
#calculate the energy of a configuration
def one_random_sweep(T):
    num_x = random.randint(0,L_x -1)
    num_y = random.randint(0,L_y-1)
    if exp(-get_delta_E(num_x,num_y)/T) > random.random():
              latt[num_x,num_y] = -latt[num_x,num_y]
def one_measure_sweep(T):
    for i in range(10):
        num_x = random.randint(0,L_x -1)
        num_y = random.randint(0,L_y-1)
        if exp(-get_delta_E(num_x,num_y)/T) > random.random():
                  latt[num_x,num_y] = -latt[num_x,num_y]


def cal_energy():
    E = 0
    for nx in range(L_x):
        for ny in range(L_y):
            nx_ml= mod(nx-1,L_x)
            nx_pl = mod(nx+1,L_x)
            ny_ml = mod(ny-1,L_y)
            ny_pl = mod(ny+1,L_y)
            E_x_y = -J*latt[nx,ny]*(latt[nx_ml,ny]+latt[nx_pl,ny]+\
            latt[nx,ny_ml]+latt[nx,ny_pl])
            E +=E_x_y
    E = E/2
    E_2 = E**2   # square of energy
    return E,E_2
```

```
start = time.time()
e_sweep = 10**5
m_sweep = 3*10**4
T_list = arange(low_T,high_T,delta_T)
M_list = zeros([len(T_list),m_sweep])
M_abs_list = zeros([len(T_list),m_sweep])
mean_M_abs =  zeros([len(T_list)])
mean_M = zeros([len(T_list)])
num_T = 0
E_list = zeros([len(T_list),m_sweep])
mean_E = zeros([len(T_list)])
E_sqrt_list = zeros([len(T_list),m_sweep])
mean_E_sqrt = zeros([len(T_list)])
M_sqrt_list = zeros([len(T_list),m_sweep])
mean_M_sqrt = zeros([len(T_list)])
M_abs_sqrt_list = zeros([len(T_list),m_sweep])
mean_M_abs_sqrt = zeros([len(T_list)])
configure = zeros([L_x,L_y])
configure_list = []
for nT in T_list:
    for nb in range(e_sweep):
        one_random_sweep(nT)
    configure_list.append(latt)
    print("doing the measurments at"+str(int(nT*1000)))
    for i in range(m_sweep):
        one_measure_sweep(nT)
        M = sum(sum(latt))/L_x/L_y   # calculate the magnetization
        M_list[num_T,i] = M
        M_abs_list[num_T,i] = abs(M)
        E_list[num_T,i],E_sqrt_list[num_T,i] = cal_energy() # find energy
        M_sqrt_list[num_T,i] = M**2   # square of magnetization
        M_abs_sqrt_list[num_T,i] = (abs(M))**2
```

```
    #calculate the mean of each observables
    mean_M[num_T] = mean(M_list[num_T,:])
    mean_M_abs[num_T] = mean(M_abs_list[num_T,:])
    mean_M_sqrt[num_T] = mean(M_sqrt_list[num_T,:])
    mean_E[num_T] = mean(E_list[num_T,:])
    mean_M_abs_sqrt[num_T] = mean(M_abs_sqrt_list[num_T,:])
    mean_E_sqrt[num_T] = mean(E_sqrt_list[num_T,:])
    num_T +=1
save("eq_configure_10_2.npy",array(configure_list))
save("mean_E_10_random_2.npy",mean_E)
save("mean_E_sqrt_10_random_2.npy", mean_E_sqrt)
c = (mean_E_sqrt-mean_E**2)/(L_x*L_y*T_list**2) #heat capacity]
save("heatcap_10_random_2.npy",c)
sus = (mean_M_abs_sqrt-mean_M_abs**2)*(L_x*L_y)/T_list
save("mean_M_sqrt_10_random_2.npy",mean_M_sqrt)
save("mean_M_10_random_2.npy",mean_M)
save("mean_M_abs_sqrt_10_random_2.npy",mean_M_abs_sqrt)
save("mean_M_abs_10_random_2.npy",mean_M_abs)
save("sus_10_random_2.npy",sus)

end = time.time()
print(end-start)

fig, ax = plt.subplots(2, 2,constrained_layout=True)
fig.suptitle('Dimension = 10x10')

ax[0, 0].plot(T_list,mean_M_abs,'ro',label = str(L_x)+"x"+str(L_y),markersize=3)
ax[0,0].set_title('Magnetization  vs temperature',fontsize = 9)
ax[0,0].set_xlabel('Temperature')
ax[0,0].set_ylabel('Magnetization')

ax[1, 0].plot(T_list,mean_E,'bo',label = str(L_x)+"x"+str(L_y),markersize=3)
```

```
ax[1,0].set_title('Average total energy vs temperature',fontsize = 9)
ax[1,0].set_xlabel("Temperature",fontsize=9)
ax[1,0].set_ylabel("average total energy",fontsize=9)



ax[0, 1].plot(T_list,sus,'go',label = str(L_x)+"x"+str(L_y),markersize=3)
ax[0,1].set_title('Susceptibility vs temperature',fontsize = 9)
ax[0,1].set_xlabel('Temperature',fontsize=9)
ax[0,1].set_ylabel("Susceptibility",fontsize=9)




ax[1, 1].plot(T_list,c,'co',label = str(L_x)+"x"+str(L_y),markersize=3)
ax[1,1].set_title('Specific heat vs temperature',fontsize = 9)
ax[1,1].set_xlabel("Temperature",fontsize=9)
ax[1,1].set_ylabel("Specific heat",fontsize=9)
plt.savefig('10x10_2_ising.png',dpi = 300)
```

B. Plot generation

```
import random
from pylab import plot,figure,errorbar,hist,subplot,title,xlabel,ylabel,legend
import matplotlib.pyplot as plt
from numpy import mod,zeros,mean,arange,sign, exp,save,load,max,where,array,log
from scipy.optimize import curve_fit

J = 1
low_T = 0.015*J      #lower temperature limit
high_T = 4.5*J    #higher temp limit
delta_T=0.015*J
T_list = arange(low_T,high_T,delta_T)
```

```
def T_c_L(x,T_c,v,x_0):
    y = T_c+(x_0*T_c)*(1/x)**(1/v)

    return y
def T_c_L_2 (x,T_c,x_0):
    y = T_c+(x_0*T_c)*(1/x)**(1/1.12)
    return y
def linear(x,a,b):
    y = a*x+b
    return y
T_list = arange(low_T,high_T,delta_T)/J

####load data
mean_E_10 = load("mean_E_10_random_2.npy")
mean_E_16 = load("mean_E_16_random_2.npy")
mean_E_24 = load("mean_E_24_random.npy")
mean_E_36 = load("mean_E_36_random_3.npy")

sus_10 = load("sus_10_random_2.npy")
sus_16 = load("sus_16_random_2.npy")
sus_24 = load("sus_24_random.npy")
sus_36 = load("sus_36_random_3.npy")

mean_M_10 = load("mean_M_abs_10_random_2.npy")
mean_M_16 = load("mean_M_abs_16_random_2.npy")
mean_M_24 = load("mean_M_abs_24_random.npy")
mean_M_36 = load("mean_M_abs_36_random_3.npy")

c_10 = load("heatcap_10_random_2.npy")
c_16 = load("heatcap_16_random_2.npy")
c_24 = load("heatcap_24_random.npy")
```

```
c_36 = load("heatcap_36_random_3.npy")
dim = array([10,16,24,36])


def problem_3():
    ##question 3

    plt.scatter(T_list,mean_E_10,s =15,label = "10x10")
    plt.scatter(T_list,mean_E_16,s =15,label = "16x16")
    plt.scatter(T_list,mean_E_24,s =15,label = "24x24")
    plt.scatter(T_list,mean_E_36,s =15,label = "36x36")
    plt.xlabel("Temperature")
    plt.ylabel("average total energy")
    plt.legend()

    plt.savefig("average_energy_plot.png",dpi = 300)
    plt.show()
#problem_3()
###question 4
def problem_4():

    plt.scatter(T_list,sus_10,s =15,label = "10x10")
    plt.scatter(T_list,sus_16,s =15,label = "16x16")
    plt.scatter(T_list,sus_24,s =15,label = "24x24")
    plt.scatter(T_list,sus_36,s = 15,label = "36x36")
    plt.xlabel("Temperature")
    plt.ylabel("susceptibility")
    plt.legend()

    plt.savefig("susceptibility_plot.png",dpi = 300)
    plt.show()
problem_4()
```

```
########## problem 4 2

def problem_4_2():
    max_sus_10 =max(sus_10)
    max_sus_16 = max(sus_16)
    max_sus_24 = max(sus_24)
    max_sus_36 = max(sus_36)
    max_sus = array([max_sus_10,max_sus_16,max_sus_24,max_sus_36])
    T_c_10 = T_list[where(sus_10 == max_sus_10)]
    T_c_16 = T_list[where(sus_16 == max_sus_16)]
    T_c_24 = T_list[where(sus_24 == max_sus_24)]
    T_c_36 = T_list[where(sus_36 == max_sus_36)]

    T_c_list = [T_c_10,T_c_16,T_c_24,T_c_36]

    print(T_c_list)
    print(max_sus)
    T_c_array = []
    for i in T_c_list:
        T_c_array.append(i[0])
    T_c_array = array(T_c_array)
    dim = array([10,16,24,36])
    plt.scatter(1/dim,T_c_list,s =15)
    plt.xlabel("$L^{-1}$")
    plt.ylabel("$T_c(L)$")
    plt.ylim([2.25,2.5])

    parameters, covariance = curve_fit(T_c_L, dim, T_c_array,maxfev = 5000)
    print(parameters)
    parameters, covariance = curve_fit(linear, 1/dim, T_c_array,maxfev = 2000)
    plt.plot(1/dim,linear(1/dim,parameters[0],parameters[1]),c ="y")
    plt.text(x = 0.06, y = 2.45,s = '$T_c(L)$ = 2.054$L^{-1}$+2.255')
```

```
    plt.savefig("Tc_l_over_l.png",dpi = 300)
    plt.show()

    print(parameters)
    #print(covariance)
problem_4_2()
###########problem 5
def problem_5():
    max_sus_10 =max(sus_10)
    max_sus_16 = max(sus_16)
    max_sus_24 = max(sus_24)
    max_sus_36 = max(sus_36)
    dim = array([10,16,24,36])
    max_sus = array([max_sus_10,max_sus_16,max_sus_24,max_sus_36])
    plt.scatter(log(dim),log(max_sus))
    parameters, covariance = curve_fit(linear, log(dim), log(max_sus))
    plt.xlabel("$log L$")
    plt.ylabel("$log \chi (T = T_c (L)$")
    plt.plot(log(dim),linear(log(dim),parameters[0],parameters[1]),"g")
    plt.text(x = 2.9, y = 2.0,s = "$log \chi (T_c (L) = 1.505log L-2.286$")
    plt.savefig("log_x_Log_L.png",dpi = 300)

    plt.show()

    print(parameters)

    #gamma_v = parameters[0]
    gamma_v = 1.504
    v = 1
    #v = 0.3
    L_x_10 = 10**(-gamma_v)*sus_10
    L_x_16 = 16**(-gamma_v)*sus_16
```

```
    L_x_24 = 24**(-gamma_v)*sus_24
    L_x_36 = 36**(-gamma_v)*sus_36
    T_c_10 = T_list[where(sus_10 == max_sus_10)]
    T_c_16 = T_list[where(sus_16 == max_sus_16)]
    T_c_24 = T_list[where(sus_24 == max_sus_24)]
    T_c_36 = T_list[where(sus_36 == max_sus_36)]

    T_c_list = [T_c_10,T_c_16,T_c_24]

    L_T_10 = 10**(1/v)*(T_list - T_c_10)
    L_T_16 = 16**(1/v)*(T_list - T_c_16)
    L_T_24 = 24**(1/v)*(T_list - T_c_24)
    L_T_36 = 36**(1/v)*(T_list- T_c_36)

    plt.scatter(L_T_10,L_x_10,s =15,label= "10x10")
    plt.scatter(L_T_16,L_x_16,s =15,label= "16x16")
    plt.scatter(L_T_24,L_x_24,s =15,label= "24x24")
    plt.scatter(L_T_36,L_x_36,s =15,label= "36x36")
    plt.xlabel("$L^{1/v}(T-T_c(L))$")
    plt.ylabel("$L^{-\gamma /v} \chi$")
    plt.legend()
    plt.savefig("scaled_sus.png", dpi = 300)
    plt.show()

#problem_5()
##############problem 6

def problem_6():
    plt.scatter(T_list,mean_M_10,s = 15,label = "10x10")
    plt.scatter(T_list,mean_M_16,s = 15,label = "16x16")
    plt.scatter(T_list,mean_M_24,s = 15,label = "24x24")
    plt.scatter(T_list,mean_M_36,s = 15,label = "36x36")
```

```
plt.legend()
plt.xlabel("Temperature")
plt.ylabel("Magnetization")
plt.savefig("magnetization_t.png",dpi = 300)
plt.show()




beta_v = 0.25
L_M_10 = 10**(beta_v)*mean_M_10
L_M_16 = 16**(beta_v)*mean_M_16
L_M_24 = 24**(beta_v)*mean_M_24
L_M_36 = 10**(beta_v)*mean_M_36
x = (L_M_10[50:200] - L_M_16[50:200])**2+(L_M_10[50:200]-L_M_16[50:200])**2+\
    (L_M_10[50:200]-L_M_24[50:200])**2+(L_M_10[50:200]-L_M_36[50:200])**2+\
    (L_M_16[50:200]-L_M_24[50:200])**2+(L_M_16[50:200]-L_M_36[50:200])**2+\
    (L_M_24[50:200]-L_M_36[50:200])**2
# plt.scatter(T_list[50:200],x)
# plt.show()



T_meet = T_list[50:200][where(x == min(x))]
print(min(x))
print(T_meet)
plt.scatter(T_list,L_M_10,s = 15,label = "10x10")
plt.scatter(T_list,L_M_16,s = 15,label = "16x16")
plt.scatter(T_list,L_M_24,s = 15,label = "24x24")
plt.scatter(T_list,L_M_36,s = 15,label = "36x36")
plt.xlabel("Temperature")
plt.ylabel("$L^{\\beta /v} M$")
```

```
    plt.legend()
    plt.savefig("magnetization_beta.png",dpi = 300)
    plt.show()

    v = 1
    T_c = 2.19
    L_T_10 = 10**(1/v)*(T_list- T_c)
    L_T_16 = 16**(1/v)*(T_list - T_c)
    L_T_24 = 24**(1/v)*(T_list - T_c)
    L_T_36 = 36**(1/v)*(T_list - T_c)
    plt.scatter(L_T_10,L_M_10,s =15,label = "10x10")
    plt.scatter(L_T_16,L_M_16,s =15,label = "16x16")
    plt.scatter(L_T_24,L_M_24,s =15,label = "24x24")
    plt.scatter(L_T_36,L_M_36,s =15,label = "36x36")
    plt.xlabel("$L^{1/v}(T-T_c')$")
    plt.ylabel("$L^{\\beta /v} M$")
    plt.legend()
    plt.savefig("magnetization_scaled.png",dpi = 300)

    plt.show()
#problem_6()

def problem_7():

    plt.scatter(T_list,c_10,s = 15,label = "10x10")
    plt.scatter(T_list,c_16,s = 15,label = "16x16")
    plt.scatter(T_list,c_24,s = 15,label = "24x24")
    plt.scatter(T_list[1:],c_36[1:],s = 15,label = "36x36")
    plt.legend()
    plt.xlabel("Temperature")
    plt.ylabel("specific heat")
    plt.savefig("specific_heat_t.png",dpi = 300)
```

```
plt.show()



v = 1
max_c_10 = max(c_10)
max_c_16 = max(c_16)
max_c_24 = max(c_24)
max_c_36 = max(c_36[1:])
print(max_c_10,max_c_16,max_c_24,max_c_36)
T_c_10 = T_list[where(c_10 == max_c_10)]
T_c_16 = T_list[where(c_16 == max_c_16)]
T_c_24 = T_list[where(c_24 == max_c_24)]
T_c_36 = T_list[where(c_36 == max_c_36)]
T_c_list = [T_c_10,T_c_16,T_c_24,T_c_36]
print(T_c_list)
T_c_array = []
for i in T_c_list:
    T_c_array.append(i[0])
T_c_array = array(T_c_array)
dim = array([10,16,24,36])

plt.scatter(dim**(-1/v),T_c_array)
parameters, covariance = curve_fit(linear, 1/dim, T_c_array,p0=[2.3,0.7])
print(parameters)
plt.plot(dim**(-1/v),linear(1/dim,parameters[0],parameters[1]),'g')
plt.ylim([2.2,2.375])
plt.xlabel("$L^{-1/v}$")
plt.ylabel("$T_c'(L)$")
plt.text(x=0.06, y = 2.28,s = "$T_c'(L) = 1.093L^{-1/v}+2.239$")
plt.savefig("T_c_prime_L", dpi = 300)
plt.show()
```

```
    max_c_list = [max_c_10,max_c_16,max_c_24,max_c_36]
    parameters, covariance = curve_fit(linear, log(dim),max_c_list, maxfev=5000)
    print(parameters)
    plt.scatter(log(dim),max_c_list)
    plt.plot(log(dim),linear(log(dim),parameters[0],parameters[1]),'orange')
    plt.xlabel("ln L")
    plt.ylabel("$C_{max}(L)$")
    plt.text(x=3, y = 1.6, s = "$c_{max}(L) = 0.379 \ln L + 0.607$")
    plt.savefig("c_max_versus_L.png", dpi = 300)
    plt.show()

problem_7()
def trape(y,T, a = 0.015):
    #trapezoidal method
    b = T
    n = y.size - 1
    h = (b-a)/n
    I = 1/2*h*(y[0]+y[-1])
    for i in range(1,n):
        I +=y[i]*h
    return I

def problem_8():
    E_list = [mean_E_10,mean_E_16,mean_E_24,mean_E_36]
    entropy_10 = trape(c_10/T_list,4.5)
    entropy_16 = trape(c_16/T_list,4.5)
    entropy_24 = trape(c_24/T_list,4.5)
    entropy_36 = trape(c_36[1:]/T_list[1:],4.5)
    c_list = [c_10,c_16,c_24]
    E_list = [mean_E_10,mean_E_16,mean_E_24]
    s = []
    for c in c_list:
```

```
        s_i = []
        for nT in T_list:
            s_i.append(trape(c/T_list,nT))
        s.append(array(s_i))
    for i in range(3):
        plt.scatter(T_list,s[i],s= 15,label = str(dim[i])+"x"+str(dim[i]))
    s_36 = []
    for nT in T_list:
        s_36.append(trape(c_36[1:]/T_list[1:],nT))
    plt.scatter(T_list[1:],s_36[1:],s = 15, label = "36x36")
    plt.legend()
    plt.xlabel("temperature")
    plt.ylabel("Entropy per spin")
    plt.savefig("ntropy_per_spin.png", dpi = 300)
    plt.show()
    print(entropy_10)
    print(entropy_16)
    print(entropy_24)
    print(entropy_36)
    for i in range(3):
        F = E_list[i]/(dim[i]**2) - T_list*s[i]
        plt.scatter(T_list,F, s = 15,label = str(dim[i])+"x"+str(dim[i]))
        print(T_list[where(F == max(F))],T_list[where(F == min(F[20:150]))])
    F_36 = mean_E_36[1:]/(36*36) - T_list[1:]*s_36[1:]
    plt.scatter(T_list[1:], F_36, s = 15,label = "36x36")
    plt.xlabel("Temperature")
    plt.ylabel("Free energy per spin")
    plt.legend()
    plt.savefig("free_energy_per_spin.png",dpi = 300)

problem_8()
```