NE 451

Assignment 4

Han Liu

20608469

Python scripts for 5-7, 5-8, 5-10 and 5-13 are attached at the end.

Python modules used in completing this homework:

import sys  
import math  
import numba  
import numpy  
import random  
import timeit  
import pickle  
import warnings  
import subprocess  
import multiprocessing  
from numba import jit  
from itertools import repeat  
import matplotlib.pyplot as plt

**Problem 5-7**

(a).

Assuming a 3D space with , a step size yielded an acceptance rate around 48% after initial 105 equilibrium trials.

*\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*

Pseudo-code:

*Initialize the polymer by random walk starting from (0, 0, 0)*

*N loop (from 10 to 100 with step size 10):*

*Timer begins*

*Monte-Carlo Loop (106 steps):*

*Polymer Loop (N monomers):*

*Select a potential new position for each monomer*

*Evaluate the new position selected for each monomer based on weight ratio*

*Accept or Reject*

*Update new positions for each monomer*

*If initial equilibrium has reached (105 Monte Carlo steps):*

*Calculate acceptance rate*

*If acceptance rate is not around 50%:*

*Throw warning to pick a new step size*

*Every 10 Monte-Carlo steps:*

*Calculate center of mass*

*Evaluate and , 104 data points*

*Timer ends*

*Outputs CPU, and for plotting*

*\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*

(a).

N vs. CPU time plot:

As expected, the CPU time increases **linearly** with the number of monomers in a polymer chain because there are more monomers whose positions need to be updated.

A close up of a mans face

Description automatically generated

N vs. double-logarithmic plot:

An entropically governed polymer has a radius of gyration of . Therefore, given . The plot below matches this equation.

A screenshot of a cell phone

Description automatically generated

N vs. double-logarithmic plot:

A close up of a mans face

Description automatically generated

(b).

vs. normalized plot with 200 bins:

The histogram is expressed as lines so that overlapping Ns can be seen.

A close up of a device

Description automatically generated

**Problem 5-8**

Assuming a 3D space with and , a step size yielded an acceptance rate around 45% after initial 106 equilibrium trials.

*\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*

Pseudo-code:

*Initialize the polymer by random walk starting from (1, 2, 3)*

*Check that in their initial positions, there isn’t any monomer that overlaps another, at least apart to start with.*

*N loop (from 10 to 100 with step size 10):*

*Timer begins*

*Monte-Carlo Loop (5 106 steps):*

*Polymer Loop (N monomers):*

*Select a potential new position for each monomer*

*Evaluate the energy for new position selected for each monomer*

*N loop:*

*Calculate the sum of LJ potential*

*Use energy and LJ potential to calculate weight ratio*

*Accept or Reject*

*Update new positions for each monomer*

*If initial equilibrium has reached (106 Monte Carlo steps):*

*Calculate acceptance rate*

*If acceptance rate is not around 50%:*

*Throw warning to pick a new step size*

*Every 10 Monte-Carlo steps:*

*Calculate center of mass*

*Evaluate and , 105 data points*

*Timer ends*

*Serialize CPU, and as output for plotting*

*Implement “Pool.map” from the Multiprocessing module to multithread different N values*

*\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*

N vs. CPU time plot:

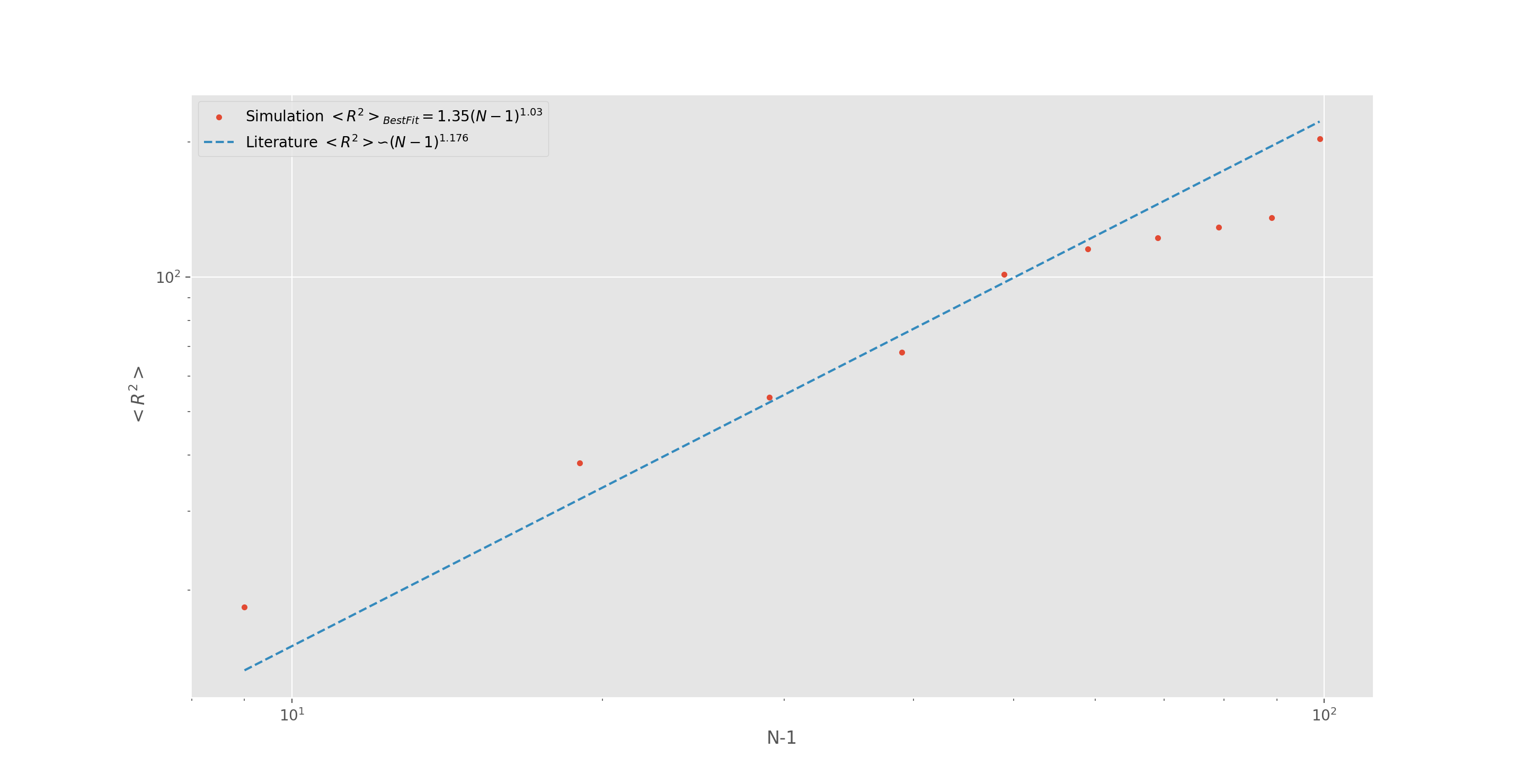
As expected, the CPU time increases **quadratically** with the number of monomers in a polymer chain because there are more monomers whose positions need to be updated and checked; in addition, for every new monomer position, the difference in Lennard Jones potential has to be calculated, too. This renders the computational complexity to be N2.

A picture containing water

Description automatically generated

N vs. double-logarithmic plot:

As evident from the plot, from the best-fit line, which is 10% off from the literature value 1.176. **Note**: since the homework was modified on 12th November to only run simulation only for N from 10 to 50, if we only find the best fit line for N = 10 to 50, then the slope becomes roughly , which is very close to the literature value 1.176.



N vs. double-logarithmic plot:

seems to increase linearly with in the double-logarithmic plot, therefore increases exponentially with N.

A picture containing racquetball

Description automatically generated

**Problem 5-10**

Assuming a 3D space with and , a step size yielded an acceptance rate around 55% after initial 105 equilibrium trials at low reduced temperature.

*\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*

Pseudo-code:

*Initialize a group of 10 gas molecules (randomly placed):*

*Check for overlapping molecules*

*Apply hard wall condition*

*Accept=0*

*For every Monte Carlo step 106:*

*For every gas molecule:*

*Access its current position*

*Calculate the LJ potential for the current position*

*Generate its new position*

*If new position is outside the wall:*

*Keep current position*

*Skip to the next gas molecule*

*Calculate the LJ potential for the new position*

*Calculate the weight ratio*

*If the weight ratio >1:*

*Accept+=1*

*Update to new position*

*If the weight ratio<1:*

*Generate a fresh random number R*

*If weight ratio>R:*

*Accept+=1*

*Update to new position*

*Else:*

*Keep the current position*

*If Monte Carlo step “s” = 105:*

*Calculate acceptance rate = accept/(s\*10)*

*If abs(acceptance rate-50%)>10%:*

*Warning(“The step size has to be adjusted”)*

*For every 10 Monte Carlo steps past the initial 105 equilibrium trials:*

*Measure and save S2gyration*

*Measure and save V*

*Measure and save V2*

*Serialize a list of V and a list of V2 for every reduced temperature value*

*Calculate <V>, <V2>,< S2gyration >*

*Calculate heat capacity*

*For the above process:*

*Implement “Pool.map” from the Multiprocessing module to multithread different values*

*Use Numba as compiler for speed boost*

*Plot vs. <V2>-<V>2 & < S2gyration >*

*\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*

vs. plot:

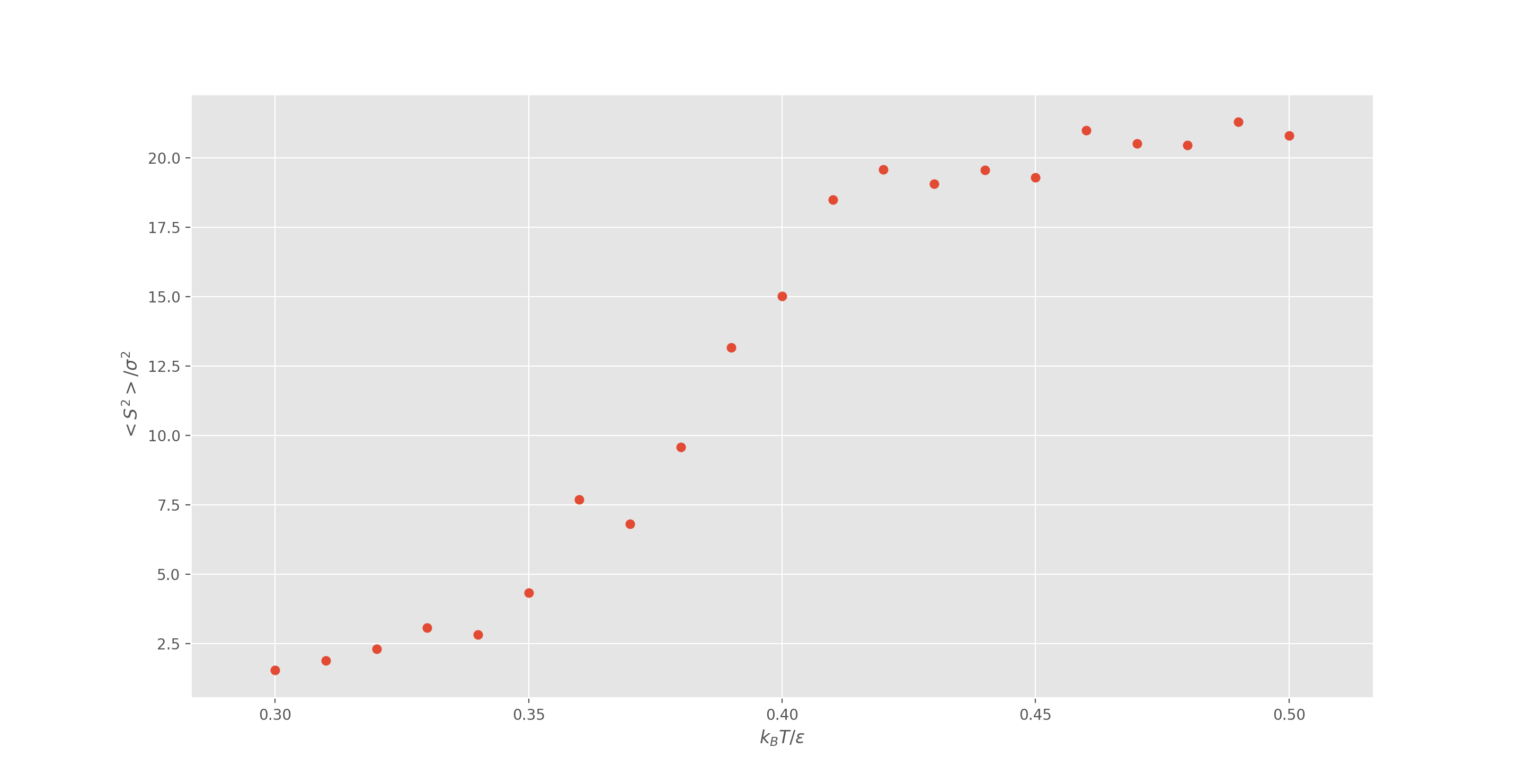
As shown below, the transition is located around 0.38 reduced temperature where the heat capacity spikes. This matches the theoretical result we obtained from the lecture.

**A picture containing wall, sky

Description automatically generated**

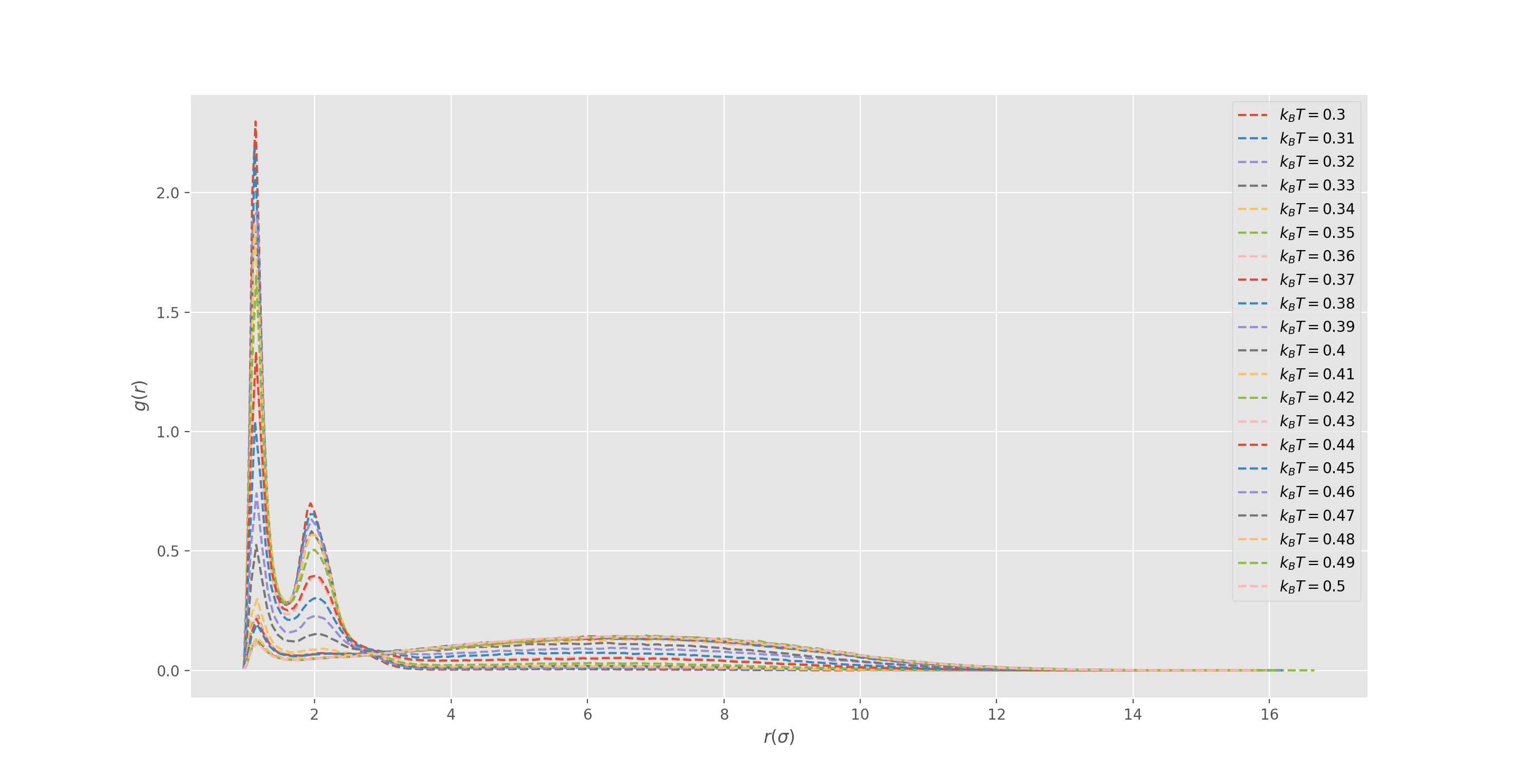
vs. plot:

As shown below, starting from reduced temperature 0.35 to 0.39, the matter undergoes a transition from its liquid to gaseous phase. This is evident from the gyration radius, which rises from low values to around. The box size is and the molecules are relatively evenly dispersed in its gaseous state; therefore, theoretically the value should converge to , which is close to the simulation result.

****

plot:

The correlation distribution shows us that as temperature increases, the distance between molecules shift its center from around to . This is expected because the gas molecules will shift towards positions with the lower energy at higher temperature values; in Lennard Jones potential, that position is situated around .



A close up of a map

Description automatically generated

**Problem 5-13**

2D Ising model with periodic boundary condition. 105 Monte Carlo steps were used.

*\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*

Pseudo-code:

*Initialize a square matrix of with randomly selected 1 or -1 entries.*

*For each Monte Carlo step:*

*Number of rejections = 0*

*For each entry ij:*

*Calculate the difference in energy*

*Calculate the weight ratio*

*If weight ratio > 1:*

*Flip the entry at ij to the opposite*

*Else:*

*Generate a random number r*

*If weight ratio > r:*

*Flip the entry at ij to the opposite*

*Else:*

*The number of rejections +=1*

*If current step is 10% of the total Monte Carlo simulation:*

*Calculate acceptance ratio*

*Calculate and store magnetization per spin m*

*Calculate and store Ising energy of the entire system E*

*Calculate and store system magnetization M*

*For the above process:*

*Implement “Pool.map” from the Multiprocessing module to multithread different values*

*Calculate , and*

*Plot against for all : , ,*

*\_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_*

vs. plot:

As expected, the per spin magnetization is uniform at low temperature and becomes more disorderly at high temperature. The more macroscopic the domain is, the more prominent this effect becomes. This is why as *L* increases, *<m>* approaches closer to 0 at high temperature.

A close up of a mans face

Description automatically generated

vs. plot:

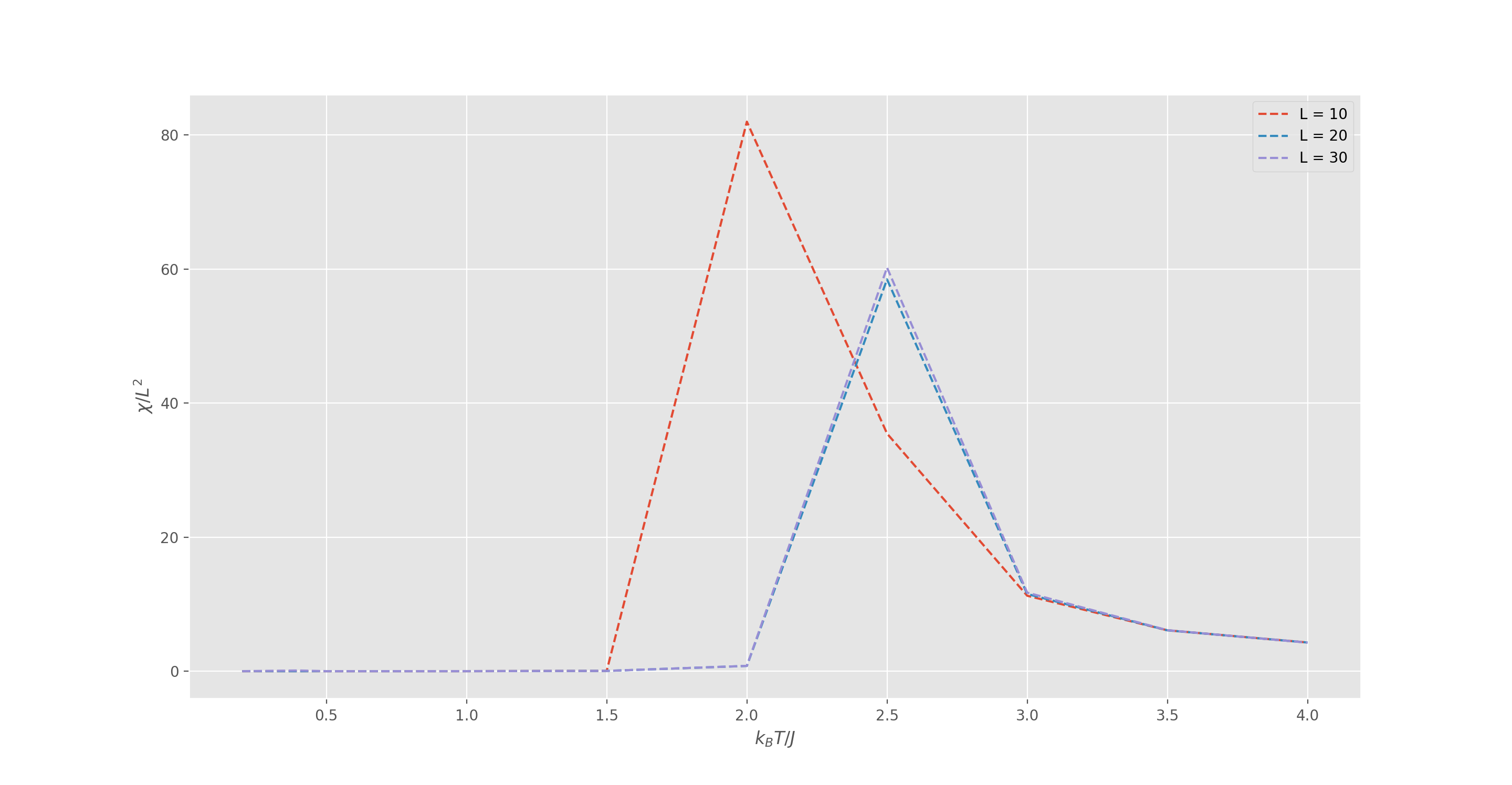
The heat capacity is proportional to variance in energy of the system. At a reduced temperature of 2.5 (matches the value from the lecture note), the phase transition takes place, this also confirm the previous figure in terms of the magnetization per spin. As *L* increases, local fluctuation has a greater possibility to appear. This is why larger *L* results in overall lower heat capacity on the system level.

A close up of a mans face

Description automatically generated

vs. plot:

The fluctuation of magnetization reaches its peak at reduced temperature 2.0 for *L = 10, 2.5* for *L = 20 & 30.*



Python script for 5-7:

#Problem 5-7  
  
class Event(object):  
 def \_\_init\_\_(self, N, M, delta, D = 3, a = 1, trial=0.1):  
 self.Polymer = Polymer(N)  
 self.M = M  
 self.delta = delta  
 self.D = D  
 self.a = a  
 self.trial = trial  
 self.S2 = []  
 self.S4 = []  
 self.Metropolis()  
  
 def Metropolis(self):  
  
 accept=0  
  
 for i in range(self.M):  
  
 n = 0  
 for j in self.Polymer.getPopulation():  
 x\_old = j.getCoordinateX()  
 dx = self.delta\*(2\*random.random()-1)  
 x\_new = x\_old+dx  
  
 y\_old = j.getCoordinateY()  
 dy = self.delta\*(2\*random.random()-1)  
 y\_new = y\_old+dy  
  
 z\_old = j.getCoordinateZ()  
 dz = self.delta\*(2\*random.random()-1)  
 z\_new = z\_old + dz  
  
 if n == 0:  
 monomer\_2=self.Polymer.getPopulation()[1]  
 x\_2start=monomer\_2.getCoordinateX()  
 y\_2start=monomer\_2.getCoordinateY()  
 z\_2start=monomer\_2.getCoordinateZ()  
  
 dE = (self.D/(2\*(self.a\*\*2)))\*((((x\_new-x\_2start)\*\*2)+((y\_new-y\_2start)\*\*2)+((z\_new-z\_2start)\*\*2))-(((x\_old-x\_2start)\*\*2)+((y\_old-y\_2start)\*\*2)+((z\_old-z\_2start)\*\*2)))  
 weightRatio = math.e\*\*((-1)\*dE)  
  
 elif n == self.Polymer.N-1:  
 monomer\_2end=self.Polymer.getPopulation()[-2]  
 x\_2end=monomer\_2end.getCoordinateX()  
 y\_2end=monomer\_2end.getCoordinateY()  
 z\_2end=monomer\_2end.getCoordinateZ()  
 dE = (self.D/(2\*(self.a\*\*2)))\*((((x\_new-x\_2end)\*\*2)+((y\_new-y\_2end)\*\*2)+((z\_new-z\_2end)\*\*2))-(((x\_old-x\_2end)\*\*2)+((y\_old-y\_2end)\*\*2)+((z\_old-z\_2end)\*\*2)))  
 weightRatio = math.e\*\*((-1)\*dE)  
  
 else:  
 monomer\_before = self.Polymer.getPopulation()[n-1]  
 x\_before = monomer\_before.getCoordinateX()  
 y\_before = monomer\_before.getCoordinateY()  
 z\_before = monomer\_before.getCoordinateZ()  
  
 monomer\_after = self.Polymer.getPopulation()[n+1]  
 x\_after = monomer\_after.getCoordinateX()  
 y\_after = monomer\_after.getCoordinateY()  
 z\_after = monomer\_after.getCoordinateZ()  
  
 dE = (self.D/(2\*(self.a\*\*2))) \* ((((x\_new-x\_before)\*\*2)+((y\_new-y\_before)\*\*2)+((z\_new-z\_before)\*\*2))-(((x\_old-x\_before)\*\*2)+((y\_old-y\_before)\*\*2)+((z\_old-z\_before)\*\*2))+(((x\_new-x\_after)\*\*2)+((y\_new-y\_after)\*\*2)+((z\_new-z\_after)\*\*2))-(((x\_old-x\_after)\*\*2)+((y\_old-y\_after)\*\*2)+((z\_old-z\_after)\*\*2)))  
 weightRatio = math.e \*\* ((-1) \* dE)  
  
 if weightRatio >= 1:  
 accept+=1  
 j.setCoordinateX(x\_new)  
 j.setCoordinateY(y\_new)  
 j.setCoordinateZ(z\_new)  
 else:  
 r=random.random()  
 if weightRatio >= r:  
 accept+=1  
 j.setCoordinateX(x\_new)  
 j.setCoordinateY(y\_new)  
 j.setCoordinateZ(z\_new)  
 n += 1  
  
 if i == int(self.trial \* self.M):  
 acceptanceRate = accept/((n+1)\*i)  
 if abs(acceptanceRate-0.5) > 0.1:  
 warnings.warn("The accpetance rate is not around 50%, please adjust step size.")  
 print(acceptanceRate)  
 sys.exit()  
 print(acceptanceRate)  
  
 if i > (self.trial \* self.M) and (i%10) == 0:  
 rc\_x=numpy.array([mono.getCoordinateX() for mono in self.Polymer.getPopulation()]).mean()  
 rc\_y=numpy.array([mono.getCoordinateY() for mono in self.Polymer.getPopulation()]).mean()  
 rc\_z=numpy.array([mono.getCoordinateZ() for mono in self.Polymer.getPopulation()]).mean()  
  
 s\_x=numpy.array([mono.getCoordinateX()-rc\_x for mono in self.Polymer.getPopulation()])\*\*2  
 s\_y=numpy.array([mono.getCoordinateY()-rc\_y for mono in self.Polymer.getPopulation()])\*\*2  
 s\_z=numpy.array([mono.getCoordinateZ()-rc\_z for mono in self.Polymer.getPopulation()])\*\*2  
 s2=(s\_x+s\_y+s\_z).mean()  
 self.S2.append(s2)  
  
 s\_x=numpy.array([mono.getCoordinateX()-rc\_x for mono in self.Polymer.getPopulation()])\*\*4  
 s\_y=numpy.array([mono.getCoordinateY()-rc\_y for mono in self.Polymer.getPopulation()])\*\*4  
 s\_z=numpy.array([mono.getCoordinateZ()-rc\_z for mono in self.Polymer.getPopulation()])\*\*4  
 s4=(s\_x+s\_y+s\_z).mean()  
 self.S4.append(s4)  
  
  
class Polymer(object):  
 def \_\_init\_\_(self,N):  
 self.N=N  
 self.population=[]  
 self.generator()  
  
 def generator(self):  
 self.population.append(Monomer(0,0,0))  
 for i in range(self.N-1):  
 x = self.population[-1].getCoordinateX() + random.choice([-1, 1])  
 y = self.population[-1].getCoordinateY() + random.choice([-1, 1])  
 z = self.population[-1].getCoordinateZ() + random.choice([-1, 1])  
 self.population.append(Monomer(x,y,z))  
  
 def getPopulation(self):  
 return self.population  
  
class Monomer(object):  
  
 def \_\_init\_\_(self,x,y,z):  
 self.x=x  
 self.y=y  
 self.z=z  
  
 def getCoordinateX(self):  
 return self.x  
  
 def setCoordinateX(self,new\_x):  
 self.x=new\_x  
  
 def getCoordinateY(self):  
 return self.y  
  
 def setCoordinateY(self,new\_y):  
 self.y=new\_y  
  
 def getCoordinateZ(self):  
 return self.z  
  
 def setCoordinateZ(self,new\_z):  
 self.z=new\_z  
  
# N = [10,20,30,40,50,60,70,80,90,100]  
# CPU = []  
# M=10\*\*6  
# delta=0.66  
#  
# for n in N:  
# start = timeit.default\_timer()  
# E=Event(n,M,delta)  
# S2=E.S2  
# S4=E.S4  
# print(n)  
# print(numpy.array(S2).mean())  
# print(numpy.array(S4).mean())  
# with open('5-7 S2 '+str(n)+' .pickle','wb') as handle:  
# pickle.dump(S2,handle)  
#  
# with open('5-7 S4 '+str(n)+' .pickle','wb') as handle:  
# pickle.dump(S4,handle)  
#  
# end= timeit.default\_timer()  
# cpu=end-start  
# CPU.append(cpu)  
#  
# with open('CPU.pickle','wb') as handle:  
# pickle.dump(CPU,handle)  
  
# N = [10,20,30,40,50,60,70,80,90,100]  
# CPU = pickle.load(open('CPU.pickle', 'rb'))  
# S2\_sq\_mean = [numpy.array(pickle.load(open('5-7 S2 '+str(n)+' .pickle', 'rb'))).mean() for n in N]  
# S4\_sq\_mean = [numpy.array(pickle.load(open('5-7 S4 '+str(n)+' .pickle', 'rb'))).mean() for n in N]  
#   
# S2=[numpy.array(pickle.load(open('5-7 S2 '+str(n)+' .pickle', 'rb'))) for n in N]  
#   
# for i in range(len(S2)):  
# S2[i]=(S2[i]\*\*0.5)/(S2\_sq\_mean[i]\*\*0.5)  
#   
# print(CPU)  
# print(S2\_sq\_mean)  
# print(S4\_sq\_mean)  
#   
# plt.style.use('ggplot')  
# fig = plt.figure()  
# ax1 = fig.add\_subplot(111)  
  
# ax1.plot(N,CPU)  
# ax1.set\_xlabel("N")  
# ax1.set\_ylabel("CPU Time (s)")  
# plt.show()  
  
# ax1.loglog(N,S2\_sq\_mean)  
# ax1.set\_xlabel("N")  
# ax1.set\_ylabel(r"$<S^{2}>/a^{2}$")  
# plt.show()  
  
# ax1.loglog(N,S4\_sq\_mean)  
# ax1.set\_xlabel("N")  
# ax1.set\_ylabel(r"$<S^{4}>/a^{4}$")  
# plt.show()  
  
# N=10  
# for i in S2:  
# G,binEdges=numpy.histogram(i,bins=180,density=True)  
# bincenters=0.5\*(binEdges[1:]+binEdges[:-1])  
# ax1.plot(bincenters,G,label="N = "+str(N))  
# ax1.set\_xlabel(r"$S/<S^{2}>^{1/2}$")  
# ax1.set\_ylabel("G(S)")  
# N+=10  
# plt.legend()  
# plt.show()

Python script for 5-8:

#Problem 5-8  
  
# specMonomer=[('x',nb.float64),('y',nb.float64),('z',nb.float64)]  
# @jitclass(specMonomer)  
class Monomer(object):  
  
 def \_\_init\_\_(self,x,y,z):  
 self.x=x  
 self.y=y  
 self.z=z  
  
 def getCoordinateX(self):  
 return self.x  
  
 def setCoordinateX(self,new\_x):  
 self.x=new\_x  
  
 def getCoordinateY(self):  
 return self.y  
  
 def setCoordinateY(self,new\_y):  
 self.y=new\_y  
  
 def getCoordinateZ(self):  
 return self.z  
  
 def setCoordinateZ(self,new\_z):  
 self.z=new\_z  
  
 def distanceSqrt(self,monomer):  
 return ((self.getCoordinateX()-monomer.getCoordinateX())\*\*2)+((self.getCoordinateY()-monomer.getCoordinateY())\*\*2)+((self.getCoordinateZ()-monomer.getCoordinateZ())\*\*2)  
  
  
class Polymer(object):  
 def \_\_init\_\_(self,N):  
 self.N=N  
 self.population=[]  
 self.generator()  
  
 def generator(self):  
 self.population.append(Monomer(1,2,3))  
 for i in range(self.N-1):  
 do=True  
 while do:  
 x = self.population[-1].getCoordinateX() + random.choice([-1, 1])  
 y = self.population[-1].getCoordinateY() + random.choice([-1, 1])  
 z = self.population[-1].getCoordinateZ() + random.choice([-1, 1])  
 potentialMonomer=Monomer(x,y,z)  
 distanceList=[]  
 for existingMonomer in self.population:  
 distanceList.append(potentialMonomer.distanceSqrt(existingMonomer))  
 checkList=[d>=1 for d in distanceList]  
 if all(checkList):  
 do=False  
 self.population.append(potentialMonomer)  
  
 def getPopulation(self):  
 return self.population  
  
class Event(object):  
 def \_\_init\_\_(self, N, M, delta, D = 3, a = 1, trial=0.1):  
 self.Polymer = Polymer(N)  
 self.M = M  
 self.delta = delta  
 self.D = D  
 self.a = a  
 self.trial = trial  
 self.S2 = []  
 self.R = []  
 self.Metropolis()  
  
 def Metropolis(self):  
 # for m in self.Polymer.getPopulation():  
 # print(str(m.getCoordinateX()) + " " + str(m.getCoordinateY()) + " " + str(m.getCoordinateZ()))  
 accept=0  
  
 for i in range(self.M):  
  
 n = 0  
 for j in self.Polymer.getPopulation():  
 x\_old = j.getCoordinateX()  
 dx = self.delta\*(2\*random.random()-1)  
 x\_new = x\_old+dx  
  
 y\_old = j.getCoordinateY()  
 dy = self.delta\*(2\*random.random()-1)  
 y\_new = y\_old+dy  
  
 z\_old = j.getCoordinateZ()  
 dz = self.delta\*(2\*random.random()-1)  
 z\_new = z\_old + dz  
  
 potentialMonomer=Monomer(x\_new,y\_new,z\_new)  
  
 if n == 0:  
 monomer\_2=self.Polymer.getPopulation()[1]  
  
 dE = (self.D/(2\*(self.a\*\*2)))\*(potentialMonomer.distanceSqrt(monomer\_2)-j.distanceSqrt(monomer\_2))  
 dP = 0  
  
 for index in range(self.Polymer.N):  
 if index!=n:  
 new\_r=potentialMonomer.distanceSqrt(self.Polymer.getPopulation()[index])  
 dP = dP + 2\*((1/(new\_r\*\*6))-(1/(new\_r\*\*3)))  
 old\_r=j.distanceSqrt(self.Polymer.getPopulation()[index])  
 dP = dP - 2\*((1/(old\_r\*\*6))-(1/(old\_r\*\*3)))  
 weightRatio = math.e\*\*((-1)\*(dE+dP))  
  
 elif n == self.Polymer.N-1:  
 monomer\_2end=self.Polymer.getPopulation()[-2]  
  
 dE = (self.D/(2\*(self.a\*\*2)))\*(potentialMonomer.distanceSqrt(monomer\_2end)-j.distanceSqrt(monomer\_2end))  
 dP = 0  
  
 for index in range(self.Polymer.N):  
 if index!=n:  
 new\_r=potentialMonomer.distanceSqrt(self.Polymer.getPopulation()[index])  
 dP = dP + 2\*((1/(new\_r\*\*6))-(1/(new\_r\*\*3)))  
 old\_r=j.distanceSqrt(self.Polymer.getPopulation()[index])  
 dP = dP - 2\*((1/(old\_r\*\*6))-(1/(old\_r\*\*3)))  
 weightRatio = math.e\*\*((-1)\*(dE+dP))  
  
 else:  
 monomer\_before = self.Polymer.getPopulation()[n-1]  
  
 monomer\_after = self.Polymer.getPopulation()[n+1]  
  
 dE = (self.D/(2\*(self.a\*\*2))) \* (potentialMonomer.distanceSqrt(monomer\_before)-j.distanceSqrt(monomer\_before)+potentialMonomer.distanceSqrt(monomer\_after)-j.distanceSqrt(monomer\_after))  
 dP = 0  
  
 for index in range(self.Polymer.N):  
 if index != n:  
 new\_r = potentialMonomer.distanceSqrt(self.Polymer.getPopulation()[index])  
 dP = dP + 2 \* ((1 / (new\_r \*\* 6)) - (1 / (new\_r \*\* 3)))  
 old\_r = j.distanceSqrt(self.Polymer.getPopulation()[index])  
 dP = dP - 2 \* ((1 / (old\_r \*\* 6)) - (1 / (old\_r \*\* 3)))  
 weightRatio = math.e \*\* ((-1) \* (dE + dP))  
  
 if weightRatio >= 1:  
 accept+=1  
 j.setCoordinateX(x\_new)  
 j.setCoordinateY(y\_new)  
 j.setCoordinateZ(z\_new)  
 else:  
 r=random.random()  
 if weightRatio >= r:  
 accept+=1  
 j.setCoordinateX(x\_new)  
 j.setCoordinateY(y\_new)  
 j.setCoordinateZ(z\_new)  
 n += 1  
  
 if i == int(self.trial \* self.M):  
 acceptanceRate = accept/((n+1)\*i)  
 if abs(acceptanceRate-0.5) > 0.1:  
 warnings.warn("The accpetance rate is not around 50%, please adjust step size.")  
 print(acceptanceRate)  
 sys.exit()  
 print(acceptanceRate)  
  
 if i > (self.trial \* self.M) and (i%10) == 0:  
 rc\_x=numpy.array([mono.getCoordinateX() for mono in self.Polymer.getPopulation()]).mean()  
 rc\_y=numpy.array([mono.getCoordinateY() for mono in self.Polymer.getPopulation()]).mean()  
 rc\_z=numpy.array([mono.getCoordinateZ() for mono in self.Polymer.getPopulation()]).mean()  
  
 s\_x=numpy.array([mono.getCoordinateX()-rc\_x for mono in self.Polymer.getPopulation()])\*\*2  
 s\_y=numpy.array([mono.getCoordinateY()-rc\_y for mono in self.Polymer.getPopulation()])\*\*2  
 s\_z=numpy.array([mono.getCoordinateZ()-rc\_z for mono in self.Polymer.getPopulation()])\*\*2  
 s2=(s\_x+s\_y+s\_z).mean()  
 self.S2.append(s2)  
  
 r\_first = self.Polymer.getPopulation()[0]  
  
 r\_last = self.Polymer.getPopulation()[-1]  
  
 r = r\_first.distanceSqrt(r\_last)  
 self.R.append(r)  
  
if 'darwin' in sys.platform:  
 print('Running \'caffeinate\' on MacOSX to prevent the system from sleeping')  
 subprocess.Popen('caffeinate')  
  
def runSim(n):  
 M = 5 \* (10 \*\* 6)  
 delta = 0.3  
 start = timeit.default\_timer()  
 E = Event(n, M, delta)  
 S2 = E.S2  
 R = E.R  
 print("Number of Monomers: " + str(n))  
 print("Expected value: "+str(n\*\*1.176))  
 print("<S2^2> = " + str(numpy.array(S2).mean()))  
 print("<R^2> = " + str(numpy.array(R).mean()))  
 with open('5-8/5-8 S2 ' + str(n) + '.pickle', 'wb') as handle:  
 pickle.dump(S2, handle)  
  
 with open('5-8/5-8 R ' + str(n) + '.pickle', 'wb') as handle:  
 pickle.dump(R, handle)  
  
 end = timeit.default\_timer()  
 cpu = end - start  
 print("CPU = " + str(cpu))  
 print("\n")  
  
 with open('5-8/5-8 CPU ' + str(n) + '.pickle', 'wb') as handle:  
 pickle.dump(cpu, handle)  
  
  
N = [10,20,30,40,50,60,70,80,90,100]  
  
if \_\_name\_\_== '\_\_main\_\_':  
 try:  
 p=multiprocessing.Pool(processes=multiprocessing.cpu\_count()-1)  
 p.map(runSim, N)  
 finally:  
 p.close()  
 p.join()  
  
N = numpy.array([10,20,30,40,50,60,70,80,90,100])  
R2 = [pickle.load(open('5-8/5-8 R ' + str(n) + '.pickle', 'rb')) for n in N]  
S2 = [pickle.load(open('5-8/5-8 S2 ' + str(n) + '.pickle','rb')) for n in N]  
R2 = [numpy.array(r2).mean() for r2 in R2]  
S2 = [numpy.array(s2).mean() for s2 in S2]  
CPU = [pickle.load(open('5-8/5-8 CPU ' + str(n) + '.pickle','rb')) for n in N]  
R2\_literature=[n\*\*1.176 for n in (N-1)]  
  
R2=numpy.array(R2)  
S2=numpy.array(S2)  
  
def best\_fit(X, Y):  
  
 xbar = sum(X)/len(X)  
 ybar = sum(Y)/len(Y)  
 n = len(X) # or len(Y)  
  
 numer = sum([xi\*yi for xi,yi in zip(X, Y)]) - n \* xbar \* ybar  
 denum = sum([xi\*\*2 for xi in X]) - n \* xbar\*\*2  
  
 b = numer / denum  
 a = ybar - b \* xbar  
  
 return a, b  
  
plt.style.use('ggplot')  
fig = plt.figure()  
ax1 = fig.add\_subplot(111)  
  
c, v=best\_fit(numpy.log10(N-1),numpy.log10(R2))  
ax1.loglog(N-1,R2,'.',label=r"Simulation $<R^{2}>\_{Best Fit} = 1.35(N-1)^{1.03}$")  
ax1.loglog(N-1,R2\_literature,'--',label=r"Literature $<R^{2}> \backsim (N-1)^{1.176}$")  
ax1.set\_xlabel("N-1")  
ax1.set\_ylabel(r"$<R^{2}>$")  
plt.legend()  
plt.show()  
  
ax1.loglog(N,S2,'--')  
ax1.set\_xlabel("N")  
ax1.set\_ylabel(r"$<S^{2}>$")  
plt.legend()  
plt.show()  
  
l=numpy.poly1d(numpy.polyfit(N,CPU,2))  
ax1.plot(N,CPU,'.',N,l(N),'--')  
ax1.set\_xlabel("N")  
ax1.set\_ylabel("CPU Time (min)")  
plt.show()

Python script for 5-10:

#Problem 5-10  
  
MC = 10\*\*6  
K = 10  
sigma = 1  
epsilon = 1  
L = 10 \* sigma  
  
reduced\_energy = numpy.linspace(0.3,0.50,21)  
  
def distanceSqrt(x,y):  
 return ((x[0]-y[0])\*\*2)+((x[1]-y[1])\*\*2)+((x[2]-y[2])\*\*2)  
  
def spawn1(N):  
 population=[(1,2,3)]  
 for i in range(N-1):  
 do=True  
 while do:  
 prev=population[-1]  
 x=prev[0]+random.choice([-1, 1])\*sigma  
 y=prev[1]+random.choice([-1, 1])\*sigma  
 z=prev[2]+random.choice([-1, 1])\*sigma  
 new=(x,y,z)  
 distanceList = []  
 for j in population:  
 distanceList.append(distanceSqrt(new,j))  
 checkList1 = [d >= 1 for d in distanceList]  
 checkList2 = [x<L,y<L,z<L]  
 checkList=checkList1+checkList2  
 if all(checkList):  
 do=False  
 population.append(new)  
 return population  
  
def spawn2(N):  
 population=[]  
 for i in range(N):  
 do=True  
 while do:  
 x=(2\*random.random()-1)\*2\*sigma  
 y=(2\*random.random()-1)\*2\*sigma  
 z=(2\*random.random()-1)\*2\*sigma  
  
 new=(x,y,z)  
 if not (new in population):  
 do=False  
 population.append(new)  
 return population  
  
def VLJSystem(l):  
 T=0  
 for i in range(len(l)):  
 t=0  
 for j in range(len(l)):  
 if i!=j:  
 d=distanceSqrt(l[i],l[j])  
 vlj=4\*epsilon\*(((sigma\*\*12)/(d\*\*6))-((sigma\*\*6)/(d\*\*3)))  
 t+=vlj  
 T+=t  
 return T  
  
def radiusGyration(l):  
 x\_c=numpy.array([i[0] for i in l]).mean()  
 y\_c=numpy.array([i[1] for i in l]).mean()  
 z\_c=numpy.array([i[2] for i in l]).mean()  
  
 centered\_one=(x\_c,y\_c,z\_c)  
 r\_g\_2 = numpy.array([distanceSqrt(i,centered\_one) for i in l]).mean()  
  
 # print(r\_g\_2)  
 return r\_g\_2  
  
  
def LJ\_Metropolis(reduced\_energy, delta, MC = MC, K = K, L = L, sigma=sigma, epsilon=epsilon):  
 population=spawn2(K)  
  
 accept=0  
  
 S2=[]  
 V\_LJ=[]  
 G = []  
  
 for mc in range(MC):  
  
 g = []  
 for i in range(K):  
  
 old\_x=population[i][0]  
 old\_y=population[i][1]  
 old\_z=population[i][2]  
  
 p1 = 0  
 for j in range(K):  
 if j!=i:  
 r1\_2=distanceSqrt(population[i],population[j])  
 g.append(r1\_2)  
 p1 = p1 + 4 \* epsilon \* (((sigma\*\*12) / (r1\_2\*\*6)) - ((sigma\*\*6) / (r1\_2\*\*3)))  
  
 n\_x=delta\*(2\*random.random()-1)  
 new\_x=old\_x+n\_x  
 n\_y=delta\*(2\*random.random()-1)  
 new\_y=old\_y+n\_y  
 n\_z=delta\*(2\*random.random()-1)  
 new\_z=old\_z+n\_z  
  
 if abs(new\_x)>=L/2 or abs(new\_y)>=L/2 or abs(new\_z)>=L/2:  
 continue  
  
 p2 = 0  
 for j in range(K):  
 if j!=i:  
 r2\_2=distanceSqrt((new\_x,new\_y,new\_z),population[j])  
 p2 = p2 + 4 \* epsilon \* (((sigma\*\*12) / (r2\_2\*\*6)) - ((sigma\*\*6) / (r2\_2\*\*3)))  
  
 if p2<p1:  
 accept+=1  
 population[i]=(new\_x,new\_y,new\_z)  
 else:  
 weight\_ratio = math.e\*\*((-1/(reduced\_energy\*epsilon))\*(p2-p1))  
 fresh\_r = random.random()  
 if weight\_ratio > fresh\_r:  
 accept+=1  
 population[i] = (new\_x, new\_y, new\_z)  
  
 if mc == int(0.1\*MC):  
 acceptanceRate = accept / (mc \* K)  
 if abs(acceptanceRate-0.5) > 0.1:  
 print("The acceptance rate is not around 50%, please adjust step size.")  
 print(acceptanceRate)  
  
 if mc > int(0.1\*MC) and mc%10==0:  
 V\_LJ.append(VLJSystem(population))  
 S2.append(radiusGyration(population))  
  
 if mc > int(0.1 \* MC) and mc % 100 == 0:  
 G = G + g  
  
 C\_MC=(((numpy.array(V\_LJ)\*\*2).mean())-(numpy.array(V\_LJ).mean()\*\*2))/(epsilon\*\*2)  
 S2\_MC=numpy.array(S2).mean()/(sigma\*\*2)  
 G = numpy.array(G)\*\*(1/2)  
  
 print("Reduced Energy: "+str(reduced\_energy)[:5])  
 print("Heat Capacity: "+str(C\_MC))  
 print("Gyration Radius: "+str(S2\_MC))  
 print('\n')  
  
 with open('5-10/C ' + str(reduced\_energy)[:5] + '.pickle', 'wb') as handle:  
 pickle.dump(C\_MC, handle)  
  
 with open('5-10/S ' + str(reduced\_energy)[:5] + '.pickle', 'wb') as handle:  
 pickle.dump(S2\_MC, handle)  
  
 with open('5-10/G ' + str(reduced\_energy)[:5] + '.pickle', 'wb') as handle:  
 pickle.dump(G, handle)  
  
reduced\_energy = numpy.linspace(0.3,0.50,21)  
delta = 0.105\*sigma  
  
  
if 'darwin' in sys.platform:  
 print('Running \'caffeinate\' on MacOSX to prevent the system from sleeping')  
 subprocess.Popen('caffeinate')  
  
if \_\_name\_\_== '\_\_main\_\_':  
 try:  
 p=multiprocessing.Pool(processes=multiprocessing.cpu\_count()-1)  
 p.starmap(LJ\_Metropolis, zip(list(reduced\_energy),repeat(delta),repeat(MC),repeat(K),repeat(L),repeat(sigma),repeat(epsilon)))  
 finally:  
 p.close()  
 p.join()  
  
reduced\_energy = numpy.linspace(0.3,0.50,21)  
  
C\_MC = [pickle.load(open('5-10/C ' + str(e)[:5] + '.pickle', 'rb')) for e in reduced\_energy]  
S2\_MC = [pickle.load(open('5-10/S ' + str(e)[:5] + '.pickle','rb')) for e in reduced\_energy]  
  
plt.style.use('ggplot')  
fig = plt.figure()  
ax1 = fig.add\_subplot(111)  
ax1.scatter(reduced\_energy,numpy.array(C\_MC)/2)  
ax1.set\_xlabel(r"$k\_{B}T/\epsilon$")  
ax1.set\_ylabel(r"$[<V^{2}>-<V>^{2}]/\epsilon^{2}$")  
plt.show()  
  
plt.style.use('ggplot')  
fig = plt.figure()  
ax1 = fig.add\_subplot(111)  
ax1.scatter(reduced\_energy,S2\_MC)  
ax1.set\_xlabel(r"$k\_{B}T/\epsilon$")  
ax1.set\_ylabel(r"$<S^{2}>/\sigma^{2}$")  
plt.show()  
  
  
plt.style.use('ggplot')  
fig = plt.figure()  
ax1 = fig.add\_subplot(111)  
for e in reduced\_energy:  
 g = pickle.load(open('5-10/G ' + str(e)[:5] + '.pickle','rb'))  
 G,binEdges=numpy.histogram(g,bins=180,density=True)  
 bincenters=0.5\*(binEdges[1:]+binEdges[:-1])  
 ax1.plot(bincenters,G,'--',label=r"$k\_{B}T = $"+str(e.round(3))[:4])  
ax1.set\_xlabel(r"$r (\sigma)$")  
ax1.set\_ylabel(r"$g(r)$")  
plt.legend()  
plt.show()

Python script for 5-13:

#Problem 5-13  
  
class IsingBox(object):  
 def \_\_init\_\_(self, L, J, n, reduced\_temperature):  
 self.L = L  
 self.J = J  
 self.n = n  
 self.reduced\_temperature = reduced\_temperature  
 self.I = []  
 self.m = []  
 self.M = []  
 self.E = []  
  
 def initialize(self):  
 for i in range(self.L):  
 self.I.append([])  
 for i in range(self.L):  
 for j in range(self.L):  
 self.I[i].append(random.choice([-1,1]))  
  
 def flip(self,i,j):  
 self.I[i][j]=(-1)\*self.I[i][j]  
  
 def magnetization(self):  
 sum=0  
 for i in self.I:  
 for j in i:  
 sum = sum + j  
 return sum  
  
 def findNeighbours(self,i,j):  
 up = [i-1, j]  
 down = [i+1, j]  
 left = [i, j-1]  
 right = [i, j+1]  
  
 if j == self.L-1:  
 right = [i, 0]  
 if i == self.L-1:  
 down = [0 , j]  
  
 l=[self.I[up[0]][up[1]], self.I[down[0]][down[1]], self.I[left[0]][left[1]], self.I[right[0]][right[1]]]  
 return l  
  
 def systemEnergy(self):  
  
 total=0  
 for i in range(self.L):  
 for j in range(self.L):  
 total = total + ((-1) \* self.J \* self.I[i][j] \* (numpy.array(self.findNeighbours(i,j)).sum()))  
  
 return total  
  
 def simulate(self):  
  
 num\_reject = 0  
  
 for mc in range(self.n):  
 for i in range(self.L):  
 for j in range(self.L):  
 dE = 2 \* self.J \* self.I[i][j] \* (numpy.array(self.findNeighbours(i,j)).sum())  
 w\_ratio = math.e\*\*((-1)\*(1/self.reduced\_temperature)\*(dE/self.J))  
 if w\_ratio >=1:  
 self.flip(i,j)  
 else:  
 r=random.random()  
 if w\_ratio>=r:  
 self.flip(i,j)  
 else:  
 num\_reject = num\_reject + 1  
  
 if int(mc/n)==0.1:  
 acceptance\_ratio = abs(num\_reject/((mc+1)\*(L\*\*2)))  
 if abs(acceptance\_ratio-0.5) > 0.1:  
 print("The acceptance rate is not around 50%, please adjust step size.")  
 print(acceptance\_ratio)  
  
 if mc > int(0.1 \* n):  
 self.m.append(abs(self.magnetization()/(self.L\*\*2)))  
 self.M.append(self.magnetization())  
 self.E.append(self.systemEnergy())  
  
  
if 'darwin' in sys.platform:  
 print('Running \'caffeinate\' on MacOSX to prevent the system from sleeping')  
 subprocess.Popen('caffeinate')  
  
n = 10\*\*5  
# n = 10\*\*6  
J = 1  
  
L = [10, 20, 30]  
  
def run(l):  
 reduced\_temperature = [0.2, 0.4, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]  
 for T in reduced\_temperature:  
 I=IsingBox(l,J,n,T)  
 I.initialize()  
 I.simulate()  
  
 print("L = " + str(l))  
 print("Reduced Temperature = " + str(T))  
  
 print("\n")  
  
 print("<m>")  
 m = numpy.array(I.m).mean()  
 with open('5-13/L' + str(l) + 'T' + str(T) +' m'+'.pickle', 'wb') as handle:  
 pickle.dump(m, handle)  
 print(m)  
  
 print("<Cv>")  
 c\_reduced = ((((numpy.array(I.E)\*\*2)/(J\*\*2)).mean())-(((numpy.array(I.E)/J).mean())\*\*2))/(l\*\*2)  
 with open('5-13/L' + str(l) + 'T' + str(T) +' C'+'.pickle', 'wb') as handle:  
 pickle.dump(c\_reduced, handle)  
 print(c\_reduced)  
  
 print("<X>")  
 chi\_reduced = (((numpy.array(I.M)\*\*2).mean())-((numpy.array(I.M).mean())\*\*2))/(l\*\*2)  
 with open('5-13/L' + str(l) + 'T' + str(T) +' X'+'.pickle', 'wb') as handle:  
 pickle.dump(chi\_reduced, handle)  
 print(chi\_reduced)  
  
 print("\n")  
  
if \_\_name\_\_== '\_\_main\_\_':  
 try:  
 p=multiprocessing.Pool(processes=multiprocessing.cpu\_count()-1)  
 p.starmap(run, zip(list(L)))  
 finally:  
 p.close()  
 p.join()  
  
reduced\_temperature = [0.2, 0.4, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0]  
  
l10 = 10  
l20 = 20  
l30 = 30  
  
m10 = [pickle.load(open('5-13/L' + str(l10) + 'T' + str(T) +' m'+'.pickle', 'rb')) for T in reduced\_temperature]  
m20 = [pickle.load(open('5-13/L' + str(l20) + 'T' + str(T) +' m'+'.pickle', 'rb')) for T in reduced\_temperature]  
m30 = [pickle.load(open('5-13/L' + str(l30) + 'T' + str(T) +' m'+'.pickle', 'rb')) for T in reduced\_temperature]  
  
c10 = [pickle.load(open('5-13/L' + str(l10) + 'T' + str(T) +' C'+'.pickle', 'rb')) for T in reduced\_temperature]  
c20 = [pickle.load(open('5-13/L' + str(l20) + 'T' + str(T) +' C'+'.pickle', 'rb')) for T in reduced\_temperature]  
c30 = [pickle.load(open('5-13/L' + str(l30) + 'T' + str(T) +' C'+'.pickle', 'rb')) for T in reduced\_temperature]  
  
chi10 = [pickle.load(open('5-13/L' + str(l10) + 'T' + str(T) +' X'+'.pickle', 'rb')) for T in reduced\_temperature]  
chi20 = [pickle.load(open('5-13/L' + str(l20) + 'T' + str(T) +' X'+'.pickle', 'rb')) for T in reduced\_temperature]  
chi30 = [pickle.load(open('5-13/L' + str(l30) + 'T' + str(T) +' X'+'.pickle', 'rb')) for T in reduced\_temperature]  
  
plt.style.use('ggplot')  
fig = plt.figure()  
ax1 = fig.add\_subplot(111)  
ax1.plot(reduced\_temperature,m10,'--',label="L = 10")  
ax1.plot(reduced\_temperature,m20,'--',label="L = 20")  
ax1.plot(reduced\_temperature,m30,'--',label="L = 30")  
ax1.set\_xlabel(r'$k\_{B}T/J$')  
ax1.set\_ylabel(r'$<m>$')  
plt.legend()  
plt.show()  
  
plt.style.use('ggplot')  
fig = plt.figure()  
ax1 = fig.add\_subplot(111)  
ax1.plot(reduced\_temperature,c10,'--',label="L = 10")  
ax1.plot(reduced\_temperature,c20,'--',label="L = 20")  
ax1.plot(reduced\_temperature,c30,'--',label="L = 30")  
ax1.set\_xlabel(r'$k\_{B}T/J$')  
ax1.set\_ylabel(r'$C\_{V}k\_{B}T^{2}/J^{2}L^{2}$')  
plt.legend()  
plt.show()  
  
plt.style.use('ggplot')  
fig = plt.figure()  
ax1 = fig.add\_subplot(111)  
ax1.plot(reduced\_temperature,chi10,'--',label="L = 10")  
ax1.plot(reduced\_temperature,chi20,'--',label="L = 20")  
ax1.plot(reduced\_temperature,chi30,'--',label="L = 30")  
ax1.set\_xlabel(r'$k\_{B}T/J$')  
ax1.set\_ylabel(r'$\chi/L^{2}$')  
plt.legend()  
plt.show()