3 MC of a Lennard-Jones system

3.1 Complete the code that the average pressure can be calculated

In this exercise, we have a Monte Carlo program for a Lennard-Jones system. To start the simulation, first of all, the system is initialized. The equilibrium properties of the system should not depend on the initial condition. For simplicity, the particles are placed in a lattice. The next step is to equilibrate the system. Predefined rounds of Monte Carlo moves are performed to read the equilibrium. Finally, one could perform more Monte Carlo moves to sample the properties of the system and this stage is named production.

The maximum displacement should be changed during the production stage so that the apriori probability is fixed to 50%. However, if one changes the maximum displacement before every move, the detailed balance is violated. It is safer to change maximum displacement a few times during the simulation.

The energy.c file contains two functions that related to the calculation of the system energy. Function EnergySystem() calculate the total system energy and viral by lopping over every particle and summing up the energy of i and i+1 particle. Function EnergyParticle() calculate the energy between two particles.

The pressure of system can be calculated by the following

$$P = \frac{\rho}{\beta} + \frac{vir}{3V} \tag{7}$$

$$vir = -r \cdot \frac{dU(r)}{dr} \tag{8}$$

In the program, the calculation of virial is missing.

```
for(j=jb;j<NumberOfParticles;j++)
{
    if (j!=i)
    {
      dr.x=pos.x-Positions[j].x;
      dr.y=pos.y-Positions[j].y;
      dr.z=pos.z-Positions[j].z;

      // apply boundary conditions
      dr.x-=Box*rint(dr.x/Box);
      dr.y-=Box*rint(dr.y/Box);
      dr.z-=Box*rint(dr.z/Box);

      r2=SQR(dr.x)+SQR(dr.y)+SQR(dr.z);

      // calculate the energy
      if(r2<SQR(CutOff))
      {
        r2i=SQR(Sigma)/r2;
        r6i=CUBE(r2i);
        Enij+=4.0*Epsilon*(SQR(r6i)-r6i);

        // start modification
        // virial
        Virij += 4.0*6*Epsilon*(2*SQR(r6i)-r6i);
        // end modification
```

```
      }
    }
}
```

3.2 Perform a simulation at $T = 2.0$ and various densities. Up to which density does the ideal gas law $\beta p = \rho$ hold?

Simulations on different density (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.2, 1.4, 1.6, 2.0, 2.2, 2.6, 3.0) and same temperature(2.0) are performed with modified run script.

```
#! /bin/sh
#
# Scriptfile
#
# rho = density
# lmax = number of steps
# nequil = number of equilibration steps
# dr = maximum displacement
# npart = number of particles

# loop over all rho
for rho in 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.2 1.4 1.6 2.0 2.2 2.6 3.0
do
cat > input <<endofdata
  ibeg nequil lmax nsamp
     0 10000 10000 1
    dr
  0.09
ndispl
    50
 npart temp rho
   500 2.0 ${rho}
 epsilon sigma mass cutoff
 1.0 1.0 1.0 3.5
endofdata
# write result to output
time ../Source/mc_nvt.exe >> output_${rho}.txt
# catch pressure and heat capacity value from output_rho(number) file
pressure="$(grep "Average Pressure" output_${rho}.txt | grep -o '[0-9].*')"
heat_capacity="$(grep "Heat capacity" output_${rho}.txt | grep -o '[0-9].*')"
# write pressure and heat capacity value to output file
printf "${rho} ${pressure} \n">>output_pressure.txt
printf "${rho} ${heat_capacity} \n">>output_cv.txt
printf "${pressure}"
printf "${heat_capacity}"
rm input
done
exit
```

Other parameters of the system are set as default. The results of the pressure and density of each simulation are written in the output_pressure.txt and plot with modified gnuplot script.

From Figure 4, we can see that the ideal gas law only holds in low density ($\rho << 1$). When the density is higher, there is much more interaction between the particle, which no longer obey

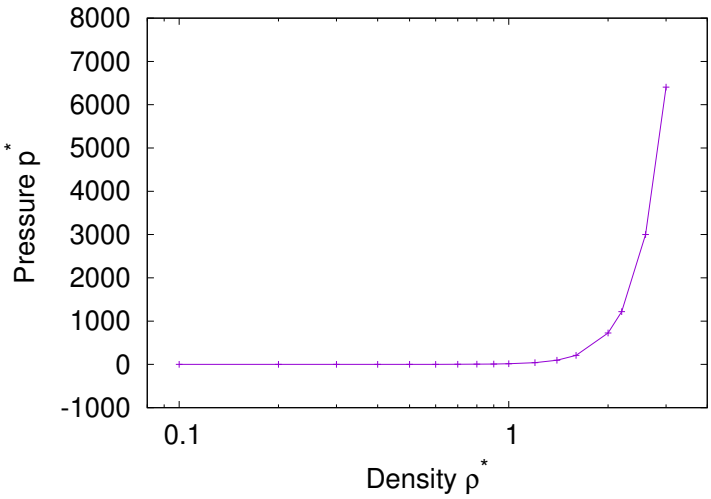

Figure 4: *Pressure of the system on different density*

Figure 5: *Visualise the state of the system with VMD*

the ideal gas behavior.

3.3 The program produces a sequence of snapshots of the state of the system. Try to visualise these snapshots using the program vmd.

To visualize the equilibrium and sampling process. Snapshots of the system are stored in pdb file. We can open the pdb file with VMD (Figure 5). The representations of the molecules can be adjusted. One can also control the speed of the animation, the visual angle, etc.

3.4 Derive a formula for the dimensionless heat capacity. Modify the program in such a way that C_v is calculated.

The heat capacity is

$$C_V = \frac{1}{N}\left(\frac{\partial U}{\partial T}\right)_V = \frac{\langle U^2\rangle - \langle U\rangle^2}{Nk_BT^2} \tag{9}$$

The unit of capacity is k_b. To make it dimensionless

$$Cv^* = \frac{C_v}{k_b} \tag{10}$$

From the equations, we know that the expectation value of the system energy and the square of system energy is needed. Similar to the calculation of pressure, we have to sample these values. Few lines are add to the function main() in mc_nvt.c.

```
if((j%SamplingFrequency)==0)
    {
        Sample(j,RunningEnergy,RunningVirial,&Pressure,FilePtr);
        PressureSum+=Pressure;
        PressureCount+=1.0;

        // start modification question 4
        // Heat capacity. You can use variables EnergySum, EnergyCount,
```

```
        // and EnergySquaredSum.
        EnergySquaredSum += SQR(RunningEnergy);
        EnergySum += RunningEnergy;
        EnergyCount += 1.0;
        // end modification
        ...
    }
```

3.5 Instead of performing a trial move in which only one particle is displaced, one can do a trial move in which all particles are displaced. Compare the maximum displacements of these moves when 50% of all displacements are accepted.

The mc_move file is slightly modified by adding a loop that all particles are displaced. Simulation is performed with the given run file. The particles number is reduced to 50. We could observe that the maximum displacements of the moves increase a lot after the modification. In the 8000/10000 step, the maximum distribution is set to 0.954. Before the modification, the maximum distribution is set to 0.224.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "system.h"
#include "ran_uniform.h"

// attempts to displace a randomly selected particle
void Mcmove(void)
{
  double EnergyNew,VirialNew,EnergyOld,VirialOld;
  VECTOR NewPosition;
  int i;

  NumberOfAttempts++;

  for(i=0;i<NumberOfParticles;i++)
  {
      // calculate old energy
      EnergyParticle(Positions[i],i,0,&EnergyOld,&VirialOld);

      // give a random displacement
      NewPosition.x=Positions[i].x+(RandomNumber()-0.5)* \
      MaximumDisplacement;
      NewPosition.y=Positions[i].y+(RandomNumber()-0.5)* \
      MaximumDisplacement;
      NewPosition.z=Positions[i].z+(RandomNumber()-0.5)* \
      MaximumDisplacement;

      // calculate new energy
      EnergyParticle(NewPosition,i,0,&EnergyNew,&VirialNew);

      if(RandomNumber()<exp(-Beta*(EnergyNew-EnergyOld)))
      {
```

```
34              // accept
35              NumberOfAcceptedMoves++;
36              RunningEnergy+=(EnergyNew-EnergyOld);
37              RunningVirial+=(VirialNew-VirialOld);
38
39              // put particle in simulation box
40  /*
41      if(NewPosition.x<0.0)
42        NewPosition.x+=Box;
43      else if(NewPosition.x>=Box)
44        NewPosition.x-=Box;
45
46      if(NewPosition.y<0.0)
47        NewPosition.y+=Box;
48      else if(NewPosition.y>=Box)
49        NewPosition.y-=Box;
50
51      if(NewPosition.z<0.0)
52        NewPosition.z+=Box;
53      else if(NewPosition.z>=Box)
54        NewPosition.z-=Box;
55  */
56
57              // update new position
58              Positions[i].x=NewPosition.x;
59              Positions[i].y=NewPosition.y;
60              Positions[i].z=NewPosition.z;
61          }
62      }
63      // choose a random particle
64      //i=NumberOfParticles*RandomNumber();
65  }
```

During the exercise session, we know that the Frenkel [1] proved that for an N-particle system,

$$P_{acc,N} = \exp(-\rho N|\Delta|) \tag{11}$$

the maximum displacement decreases with N, the number of particles displaced at a time. The low maximum displacement means that the phase space is slowly sampled. So it is inefficient that doing a trial move in which all particles are displaced.

3.6 Instead of using a uniformly distributed displacement, one can also use a Gaussian displacement. Does this increase the efficiency of the simulation?

If one uses a Gaussian displacement, on the one hand, there is more chance for a small move and a small move is more likely to be accepted. But the small move will contribute less to the efficiency of the sampling.

On the other hand, there is less chance for a large move. Although a large move is less likely to be accepted, it contributes more to the efficiency of the sampling. The two effects may compensate each other and finally may not increase the efficiency of the simulation.