NB: The graded, first version of the report must be returned if you hand in a second time!

H1b: MD Simulation - Dynamic Properties

Navid Mousavi - Daniel Cole - Nils Tornberg

November 29, 2020

Task №	Points	Avail. points
$\sum$		

Introduction

The following is the report of the HW1 on the study of Aluminum static properties using molecular dynamic techniques.

Task 1

This task was dealing with the initialization and using the provided help routines. Following the instructions, the function in H1lattice.c is used and an FCC aluminum lattice containing 256 atoms is created, which is equivalent to a $4 \times 4 \times 4$ supercell. We looked at the different potential energies by varying the lattice constant a_0 in the interval $[4.0, 4.09]$(Å). The result is presented in Fig.1, which is consistent with Fig. 1. of the problem description document. The minimum energy occurs at $a_0 = 4.03$Å.

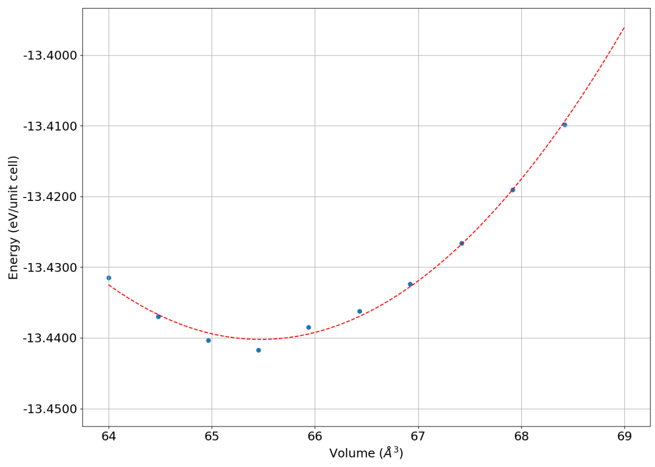
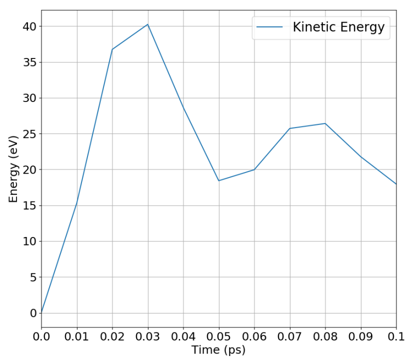


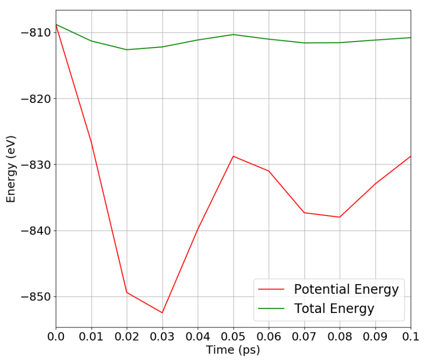
Figure 1: Potential energy (eV) against primitive cell volume ($Å^3$). Red line shows the quadratic fitting.

Task 2

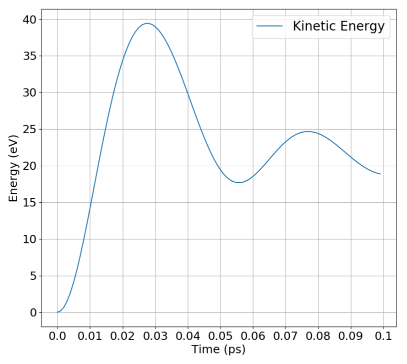
In this task the initial positions of atoms is displaced in the lattice using uniform random noise with magnitude of 6.5% of a_0, and evolved the system using Velocity-Verlet algorithm till it reaches equilibrium. Choosing the correct time step is crucial, since it has to be large enough to sample as much of phase space as possible and in the meantime small enough to generate accurate trajectories [MD-lecture-notes]. One important measure would be not violating the total energy conservation and not drifting in the long run. We checked for the short time conservation which is presented in Fig2 for $dt = 0.01ps$ and $dt = 0.001ps$. As it is expected, in the beginning energy is totally potential and very soon it is be distributed between potential and kinetic types.
Based on the observation, $dt = 0.001(ps)$ is used for the simulation. To check there exists no drift in the long time simulation in the energies, the conservation after long run $T = 1ps$ and $T = 10ps$ were studied and the result is shown in Fig.3, which show the stability of Velocity-Verlet algorithm in the long time simulations. The average temperature was calculated using relation (49) in MD lecture notes, using Boltzmann constant $k_B = 8.61733034 \times 10^{-5} eV/K$ and the temperature was calculated to be $744.376K$



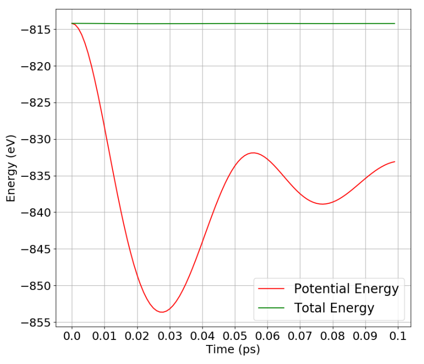
(a)



(b)

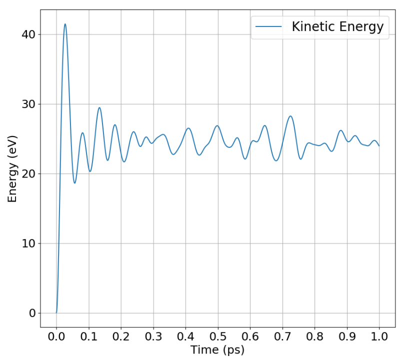


(c)

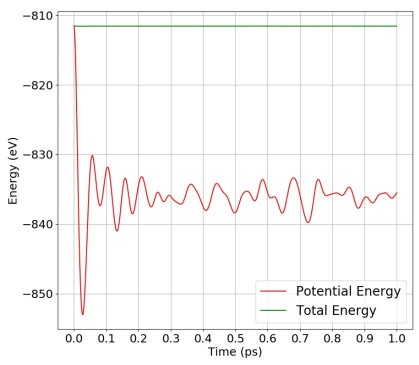


(d)

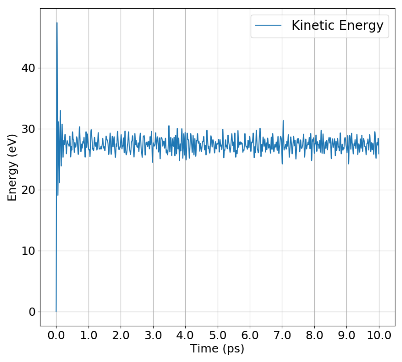
Figure 2: Comparison of different time step sizes dt on the conservation of total energy in the short time scales. (a) and (b) correspond to $dt = 0.01$ and (c), and (d) are the energy plots for $dt = 0.001$. (d) shows the negligible oscillation in total energy.



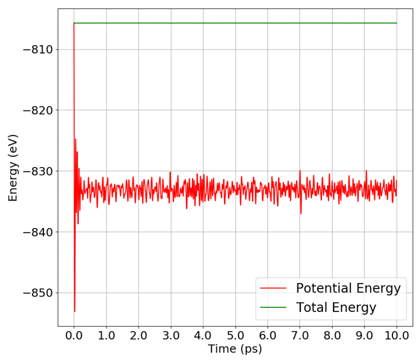
(a)



(b)



(c)



(d)

Figure 3: Checking the conservation of total energy in long run. (a), and(b) show the energies for $T = 1ps$, and (c), and (d) correspond to $T = 10ps$. The conservation of total energy is clearly preserved.

Task 3

For this task we were supposed to equilibrate the system to temperature $500°C$ and pressure equal to $1bar$. For this purpose one can scale the velocity and position using relations in appendix E of the MD lecture notes, by comparing the instantaneous values for pressure and temperature with desired values T_{eq}, P_{eq}, at each time step. To have consistent units for pressure GPa is used as unit and calculated pressures during simulation in the asu were multiplied by 160.2 to convert. Therefor, $P_{eq} = 1bar = 10^{-4}GPa$. Also, the units for temperature were K in the simulation.

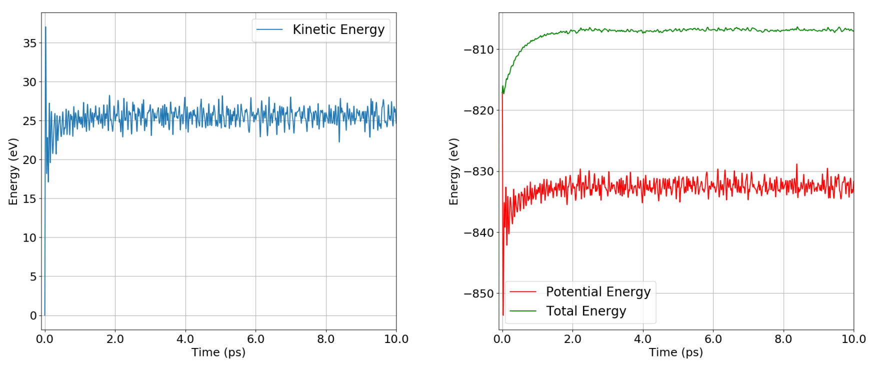
In order to do the equilibration, τ is set equal to $500dt$ and equlibration is continued for $10,000$ time steps (200τ), with isothermal compressibility $\kappa_T = 0.01385GPa$. System was successfully equlibrated in the pressure and temperature in question. Fig.4 shows the energies behavior during equlibration phase, as it is expected the total energy is not constant and varies due to scaling until the system reaches desired temperature and pressure.

Temperature behavior during equlibration is plotted in Fig.5, which shows the convergence to $T_{eq} = 773.15K$.

Also, to equilibrate pressure, positions were scaled as it is described in appendix E of MD lecture notes. Fig.6a shows the pressure convergence to $P_{eq} = 1bar$. Since the fluctuation in pressure is very large it is hard to conclude that system has equlibrated, therefor the lattice constant a_0 was studied which is shown in Fig.6b. This ensures that the system has converged to the asked pressure $P_{eq} = 1bar$. The lattice constant is found to be 4.088Å.

In addition, to make sure that the system stayed in the solid state during equilibration, the X coordinate of position of 5 selected atoms is plotted in Fig.7. It is evident from the figure that atoms only oscillate around their equilibrium position, which admits that system has been in solid state during equlibration.

After equlibrating in $P_{eq} = 1bar$ and $T_{eq} = 773.15$, the system were studied for another $10ps$ to calculate the average values. Fig.8 represents the energy behavior during equilibrium, where you can see the total energy is conserved and fluctuations happen in the kinetic and potential energies. The values found for system variables are $\langle T\rangle = 774.09K$, $\langle P\rangle = 0.0005GPa$, $\langle a_0\rangle = 4.09$Å, $\langle E_{kin}\rangle = 25.54eV$. $\langle E_{pot}\rangle = -832.56eV$, and $E = -807.02eV$.



(a) (b)

Figure 4: Energies during equilibration phase for task 3 (solid Al). (a) shows the kinetic and (b) potential and total energies. Total energy is not constant as expected during equilibration due to scaling position and velocities.

Task 4

For this task the procedure followed in the previous task were repeated for temperature $T_{eq} = 973.15K$, which is higher than melting point of Al. The only difference was in order to make sure the system is melted and it reaches the

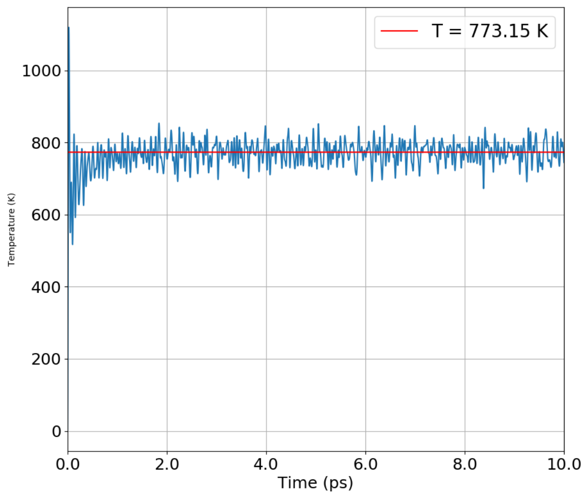


Figure 5: Temperature convergence to $T_{eq} = 773.15K$ during equilibration phase.

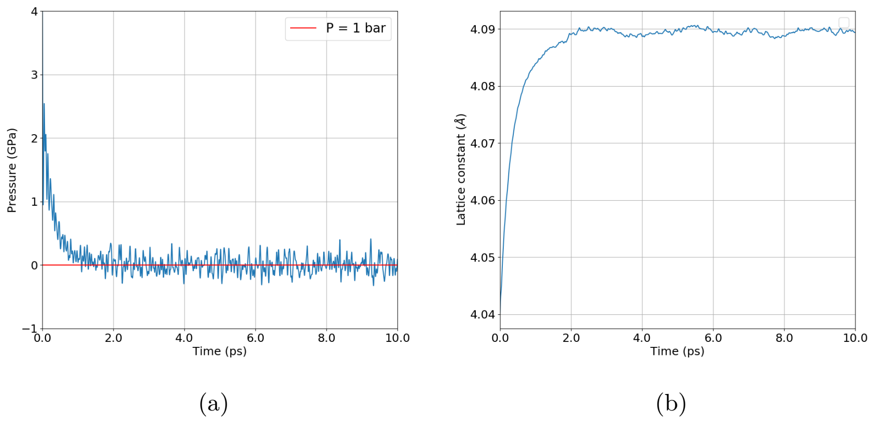


Figure 6: (a) Pressure and (b) lattice constant evolution during equilibration phase.

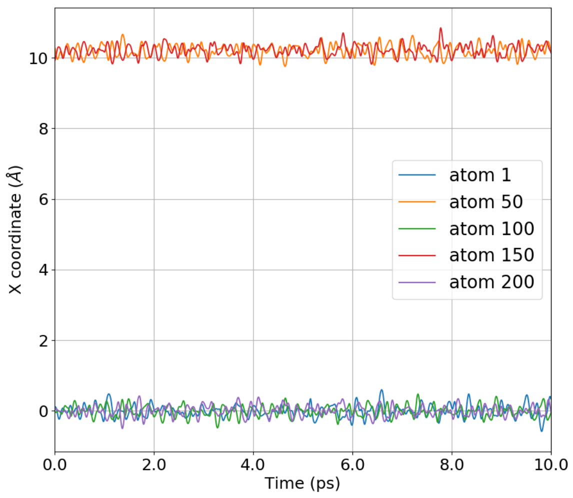


Figure 7: Trajectory of 5 selected atoms during equlibrating, which shows the system is remained in the solid state and atoms only oscillate around their equilibrium point.

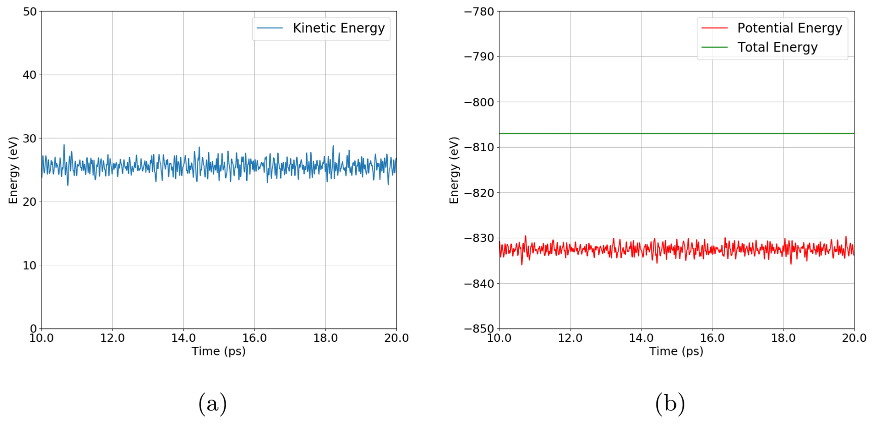


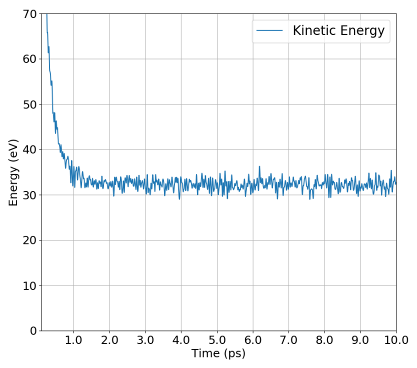
Figure 8: Energies after equilibration phase for task 3 (solid Al). (a) shows the kinetic and (b) potential and total energies. Total energy is conserved as expected during equilibrium.

liquid state, a higher initial potential energy is used with displacing atoms up to 15% of the lattice constant a_0, and then started to decrease the temperature. All other parameters are same as task3.

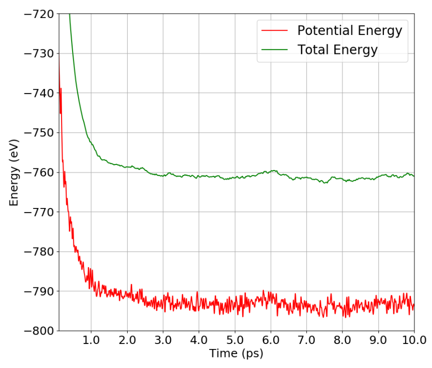
Energies evolution during equilibration is illustrated in Fig.9. Temperature is presented in Fig.10, pressure and lattice constant a_0 in Fig.11a and 11b. X coordinate for 5 selected atoms during equilibration is plotted in Fig.12, which shows that system is in liquid state and atoms can move more freely compared to task 3 (solid state).

Finally, after equilibration the system were again studied for another $10ps$ to calculate the average quantities and Fig.13 illustrates the energy plots in the equilibrium state, where total energy is constant.

Average quantities calculated are $\langle T \rangle = 984.39K$, $\langle P \rangle = 0.052GPa$, $\langle a_0 \rangle = 4.26\text{Å}$, $\langle E_{kin} \rangle = 32.45eV$. $\langle E_{pot} \rangle = -794.06eV$, and $E = -761.61eV$.



(a)



(b)

Figure 9: Energies during equilibration phase for task 4 (liquid Al). (a) shows the kinetic and (b) potential and total energies. Total energy is not constant as expected during equilibration due to scaling position and velocities.

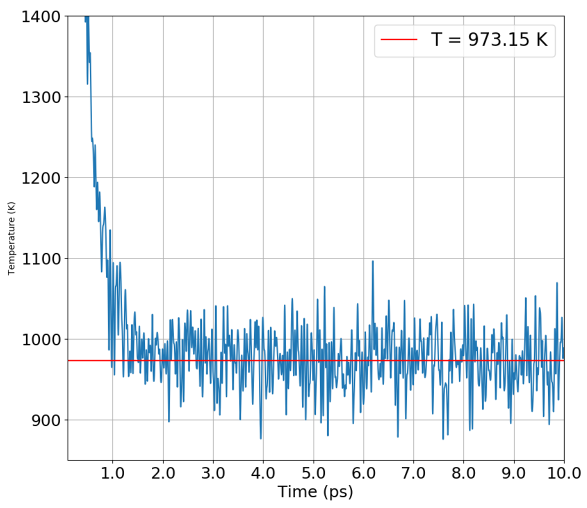
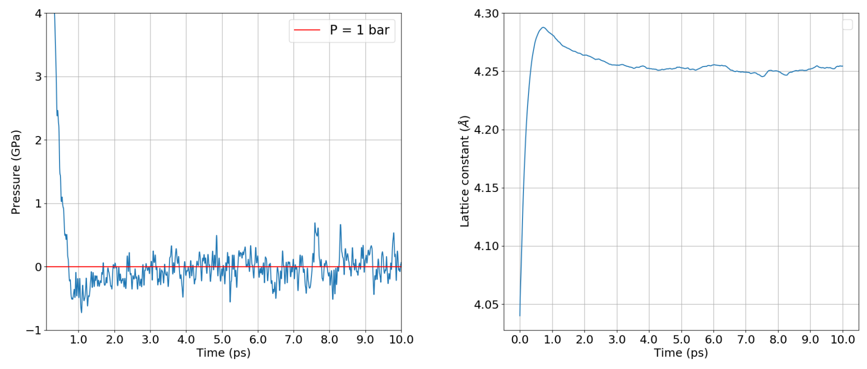


Figure 10: Temperature convergence to $T_{eq} = 973.15K$ during equilibration phase.



(a) (b)

Figure 11: (a) Pressure and (b) lattice constant evolution during equilibration phase.

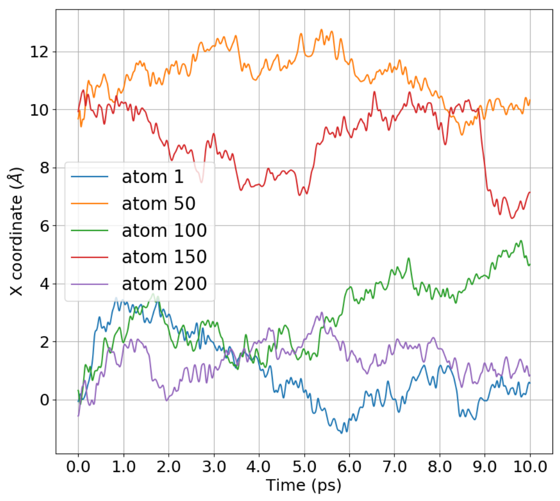
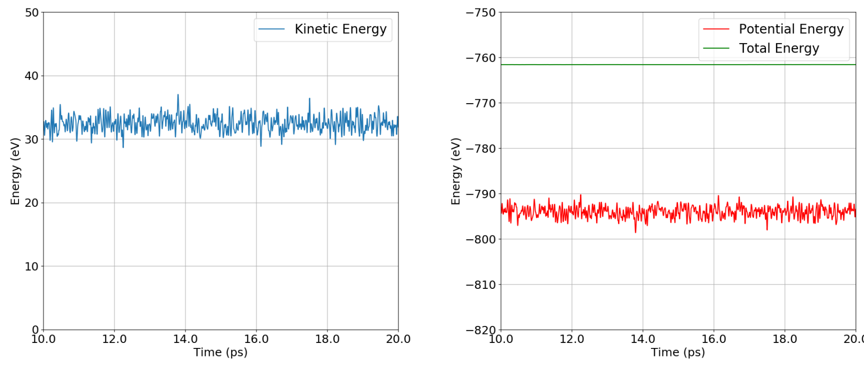


Figure 12: Trajectory of 5 selected atoms during equlibrating, which shows the system is in liquid state and atoms move more freely in the space.



(a) (b)

Figure 13: Energies after equilibration phase for task 3 (solid Al). (a) shows the kinetic and (b) potential and total energies. Total energy is conserved as expected during equilibrium.

A Source Code

I did not make any changes to H1potential.c and H1lattice.c routine files, then I did not include them here.

A.1 main.c

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include "H1lattice.h"
#include "H1potential.h"
#include "vv.h"

/* Main program */
int main()
{
  /*setting the random seed to time*/
  srand(time(NULL));
  double random_value;
  double mag;

  /* defining the necessary variables */
  /* [L]:angstrom - [T]:ps - [M]:m_asu - [Pres.]:GPa - [Temp.]:K*/
  int Nc = 4, N = 256 , n_timesteps = 10000;
  double (*pos)[3] = malloc(sizeof(double [N][3]));
  double *potE = malloc(sizeof(double) * n_timesteps);
  double *kinE = malloc(sizeof(double) * n_timesteps);
  double *totE = malloc(sizeof(double) * n_timesteps);
  double *Pressure = malloc(sizeof(double) * n_timesteps);
  double *Temperature = malloc(sizeof(double) * n_timesteps);
  double (*check_particles)[5] = malloc(sizeof(double [n_timesteps][5]));
  double *distances = malloc(sizeof(double) * (N*(N-1)/2.0));
  double (*v)[3] = malloc(sizeof(double [N][3]));
  double a0 = 4.04 , dt = 0.001 , tau = 500*dt , m = 0.002796;
  double L = Nc*a0 , T_eq = 700.0+273.15 , P_eq = 0.0001;
  /*L:lattice size , T_eq: [K], P_e:[GPa] */
  double k_b = 0.0000861733; /*Boltzamnn constant [eV/K] */


  /* Initialize the positions*/
  init_fcc(pos, Nc, a0);

  /* adding noise to positions */
  for (int i = 0; i < N; ++i)
    {
      for (int j = 0; j < 3; ++j)
    {
      mag = a0*0.065; /* setting the magnitude of the maximum noise to 6.5% of ↩
           the position value */
      random_value = ((double) rand() / (double) RAND_MAX)*(2.0*mag) - mag;
      pos[i][j] += random_value;
    }
    }

  /* Initializing velocity*/
  for (int i = 0; i < N; ++i) {
    for (int j = 0; j < 3; ++j) {
      v[i][j] = 0.0;
    }
  }

  /* Calling velocity_verlet to calculate the time evolution of the system*/
  velocity_verlet(n_timesteps , N , v , pos , dt , m , L , potE , kinE , totE ,
          Pressure , Temperature , k_b , T_eq , P_eq, tau , check_particles);

  /* writing to file the energies, pressure, temperature*/
  write_to_file("energies.txt", kinE , potE , totE , Pressure , Temperature ,  ↩
       n_timesteps);

  /*wrting to file the positions to check the solid or liquid phase*/
  FILE *f = fopen("positions.txt", "w");
  for (int i = 0; i < n_timesteps ; ++i) {
    for (int j = 0 ; j < 5; ++j) {
      fprintf(f, "%f\t", check_particles[i][j]);
    }
    fprintf(f, "\n");
  }
  fclose(f);

  /* calling the pair distance calculator and writing the result to file  */
  pair_distance(pos , distances , N);
  FILE *g = fopen("pair-dist.txt", "w");
  for (int i = 0; i < N*(N-1)/2 ; ++i) {
    fprintf(g, "%f\n", distances[i]);
  }
  fclose(g);


  /*releasing the memory*/
  free(pos); pos = NULL; free(potE) ; potE = NULL; free(kinE); kinE = NULL;
  free(totE); totE = NULL; free(v) ; v = NULL;
  free(check_particles); check_particles= NULL; free(Temperature); Temperature= NULL;
  free(Pressure); Pressure = NULL; free(distances); distances = NULL;


  return 0;
}
```

A.2 velocity-verlet function and other functions

```c
/*
File containing the functions velocity_verlet,
pair_distance calculator and write_to_file used in the main
*/

#include <stdio.h>
#include <stdlib.h>
#include "H1potential.h"
#include <math.h>


/* function responsible for equilibrating Temperature and Pressure*/
void equilibrate(double pos[][3] , double v[][3], double P_eq , double T_eq ,
         double T ,double P,double tau , double dt , int n_particles, double *L)
{
  double alpha_T , alpha_P;
  /*scaling the velocity*/
  alpha_T = 1+(2.0*dt/tau)*((T_eq - T)/T);
  for (int i = 0 ; i < n_particles; ++i)
    {
      for (int j = 0; j<3; ++j)
    {
      v[i][j] *= sqrt(alpha_T);
    }
    }
  /*scaling position*/
  alpha_P = 1-(0.01385*dt/tau*(P_eq - P));
  for (int i = 0 ; i < n_particles; ++i)
    {
      for (int j = 0; j < 3; ++j)
    {
      pos[i][j] *= cbrt(alpha_P);
    }
    }
  *L *= cbrt(alpha_P);

}

/*velocity_verlet function to calculate the time evolution of the systme*/
void velocity_verlet(int n_timesteps, int n_particles, double v[][3], double pos[][3],
             double dt, double m , double L , double *potE , double *kinE ,
             double *totE , double *Pressure , double *Temperature , double k_b ,
             double T_eq , double P_eq , double tau , double check_particles[][5])
{

  double s1 , s2;
  /*array to store forces on each particle*/
  double (*forces)[3] = malloc(sizeof(double [n_particles][3]));
  /*calling potential function*/
  potE[0] = get_energy_AL(pos , L , n_particles);
  kinE[0] = 0.0;
  totE[0] = potE[0];
  /*
    Initializing the check_particles with the initial positions of selected atoms
    atoms 0,50,100,150,200
  */
  check_particles[0][0] = pos[0][0];
  check_particles[0][1] = pos[50][0];
  check_particles[0][2] = pos[100][0];
  check_particles[0][3] = pos[150][0];
  check_particles[0][4] = pos[200][0];

  get_forces_AL(forces , pos , L , n_particles);
  /* Time step loop*/
  for (int t = 1; t < n_timesteps + 1; ++t)
```

```c
    {
      /* v(t+dt/2) */
      for (int i = 0; i < n_particles; i++)
    {
      for (int j = 0 ; j < 3; ++j) {
        v[i][j] += dt * 0.5 * forces[i][j]/m;
      }
    }
      /* q(t+dt) */
      for (int i = 0; i < n_particles ; ++i)
    {
      for (int j = 0; j < 3; ++j) {
        pos[i][j] += dt * v[i][j];
      }
    }
      /* a(t+dt) */
      get_forces_AL(forces , pos , L , n_particles);

      /* v(t+dt) */
      for (int i = 0; i < n_particles ; ++i)
    {
      for (int j = 0; j < 3; ++j) {
        v[i][j] += dt * 0.5 * forces[i][j]/m;
      }
    }
      /*new potential */
      potE[t] = get_energy_AL(pos , L , n_particles);

      /* sum(v^2) */
      s2 = 0.0;
      for (int i = 0; i < n_particles; ++i) {
    s1 = 0.0;
    for (int j = 0; j < 3; ++j) {
      s1 += v[i][j]*v[i][j];
    }
    s2 += s1;
      }
      /*kinetic energy*/
      kinE[t] = 0.5*s2*m;
      totE[t] = potE[t] + kinE[t];

      /*calling virial to calculate Pressure*/
      double W= get_virial_AL(pos , L , n_particles);
      Pressure[t] = (1.0/(L*L*L))*(W+m*s2/3.0)*(160.2) /* in GPa units */;
      printf("%f\n", L*L*L); /*printing volume*/
      /*instantaneous Temperature*/
      Temperature[t] = 2.0/(3*n_particles*k_b)*kinE[t];
      /*equilibrate if time step was less than 10tau*/
      if(t < 5000){
    equilibrate(pos , v , P_eq , T_eq ,Temperature[t] , Pressure[t] ,  tau , dt , n_particles, &L);
      }
      /*updating the check_particles*/
      check_particles[t][0] = pos[0][0];
      check_particles[t][1] = pos[50][0];
      check_particles[t][2] = pos[100][0];
      check_particles[t][3] = pos[150][0];
      check_particles[t][4] = pos[200][0];
    }
  /*releasing memory*/
  free(forces); forces = NULL;

}



/*write_to_file function to print the energies and pressure and temperature */
void write_to_file(char *fname,  double *kinE , double *potE ,
           double *totE , double *Pressure ,
           double *Temperature , int n_timesteps)
{
  FILE *fp = fopen(fname, "w");
  for(int i = 0; i < n_timesteps; ++i){
    fprintf(fp, "%f\t%f\t%f\t%f\t%f\n", kinE[i] , potE[i] , totE[i], Pressure[i], Temperature[i]);
  }
  fclose(fp);
}

/* calculating the pair distance for all pairs of atoms and saving in distances */
void pair_distance(double pos[][3], double *distances, int n_particles)
{
  double d;
  int ind = 0; //index to pair (N(N-1)/2 pairs)
  for (int i = 0 ; i < n_particles; ++i)
    {
      for (int j = i+1; j < n_particles; ++j)
    {
      d = 0.0;
      for (int k = 0 ; k < 3; ++k)
```

```
        {
          d += (pos[i][k]-pos[j][k])*(pos[i][k]-pos[j][k]);
        }
      distances[ind] = sqrt(d);
      ind++;
    }
    }
}
```

A.3 plotting

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 21 13:08:59 2020

@author: navid
"""
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import FormatStrFormatter

#Plotting task 1 energy vs vol
d = np.loadtxt("energy.txt")

x = d[:,0]**3
y = d[:,1]/64
z = np.polyfit(x,y,2)
p = np.poly1d(z)

fig, ax = plt.subplots()

ax.xaxis.set_major_formatter(FormatStrFormatter('%.0f'))
ax.yaxis.set_major_formatter(FormatStrFormatter('%.4f'))
plt.scatter(x, y)
plt.xlabel(r'Volume ($\AA^3$)', fontsize=18)
plt.ylabel('Energy (eV/unit cell)',fontsize=18)
plt.plot(np.linspace(64,69),p(np.linspace(64,69)), 'r--' )
plt.tick_params(axis='both', which='major', labelsize=18)
plt.tick_params(axis='both', which='minor', labelsize=18)
plt.grid()
plt.savefig('task1.png')


#plotting energies
d = np.loadtxt("energies.txt")

plt.figure(figsize=(10,9))
t = len(d[:,0])
n_points = t/10
dt = 0.001
plt.plot(d[:,0], label = 'Kinetic Energy')
plt.plot(d[:,1], label = 'Potential Energy')
plt.plot(d[:,2], label = 'Total Energy')
plt.legend(prop={"size":20})
plt.grid()
plt.xticks(np.arange(0,t+1,n_points),np.arange(0,t+1,n_points)*dt)
plt.xlabel(r'Time (ps)', fontsize=18)
plt.ylabel(r'Energy (eV)', fontsize=18)

plt.tick_params(axis='both', which='major', labelsize=18)
plt.tick_params(axis='both', which='minor', labelsize=18)
plt.savefig('energies.png')

#calculating the average temperature
kinE_average = np.average(d[5000:,0])
k_b = 8.61733034*10**-5
N = 256
T = (2/(3*N -3 ))*(kinE_average)/(k_b)
print(f"average temperature is : {T}")


#calculating heat capacity
kinE = d[5000:,0]
kinE_average = np.average(kinE)
kinE_2 = (kinE)**2
kinE_2_avg = np.average(kinE_2)
fluc = kinE_2_avg - kinE_average**2
cv = (3/2*N*k_b)/(1-(2/(3*N*k_b**2*(973.15**2))*fluc))
print(f'C_v = {}')

#plotting pressure
plt.figure(figsize=(10,9))
pressure = d[:,3]
plt.plot(pressure)
plt.ylabel(r'Pressure (GPa)', fontsize=18)
```

```python
plt.xlabel('Time (ps)', fontsize=18)
t = len(d[:,4])
n_points = t/10
dt = 0.001
plt.xticks(np.arange(0,t+1,n_points),np.arange(0,t+1,n_points)*dt)
plt.grid()
plt.tick_params(axis='both', which='major', labelsize=18)
plt.tick_params(axis='both', which='minor', labelsize=18)
plt.savefig('pressure.png')

#plotting temperature
plt.figure(figsize=(10,9))
x = np.ones((len(d[:,4])))
temperature = d[:,4]
plt.plot(temperature)
plt.plot(x*773.15, color = 'red' , label='T = 773.15 K')
plt.ylabel(r'Temperature (K)')
plt.xlabel('Time (ps)', fontsize=18)
t = len(d[:,4])
n_points = t/10
dt = 0.001
plt.xticks(np.arange(0,t+1,n_points),np.arange(0,t+1,n_points)*dt)
plt.grid()
plt.legend(prop={"size":20})
plt.tick_params(axis='both', which='major', labelsize=18)
plt.tick_params(axis='both', which='minor', labelsize=18)
plt.savefig('temp.png')



#plotting check positions of 5 selected atoms
plt.figure(figsize=(10,9))
pos = np.loadtxt('positions.txt')
plt.plot(pos[:,0], label = 'atom 1')
plt.plot(pos[:,1], label = 'atom 50')
plt.plot(pos[:,2], label = 'atom 100')
plt.plot(pos[:,3], label = 'atom 150')
plt.plot(pos[:,4], label = 'atom 200')
plt.legend(prop={"size":20})
plt.grid()
plt.xticks(np.arange(0,t+1,n_points),np.arange(0,t+1,n_points)*dt)
plt.xlabel(r'Time (ps)', fontsize=18)
plt.ylabel(r'X coordinate ($\AA$)', fontsize=18)
plt.tick_params(axis='both', which='major', labelsize=18)
plt.tick_params(axis='both', which='minor', labelsize=18)
plt.savefig('positions.png')



#plotting the histogram of N<r>
plt.figure(figsize=(10,9))
pair = np.loadtxt('pair-dist.txt')
plt.hist(pair,100)
plt.tick_params(axis='both', which='major', labelsize=18)
plt.tick_params(axis='both', which='minor', labelsize=18)
plt.xlabel(r'pair distance ($\AA$)', fontsize=18)
plt.ylabel('number of pairs', fontsize=18)


#calculating the N_ideal
k = np.linspace(1,53)
#delta_r = 0.5152963499999998
delta_r = 5.1529635
n_ideal = (N-1)/(4*4.04)**3*(4*np.pi/3)*(3*k**2-3*k+1)*delta_r**3

#calculating g(r)
p = plt.hist(pair,normed=True)
n = plt.hist(n_ideal,normed=True)
g = p[1]/n[1]
plt.hist(g)
```