O. Abdurazakov
NC State *University, Department of Physics, Raleigh, NC 27695*

Molecular Dynamics of Lennard-Jones Particles in 2D

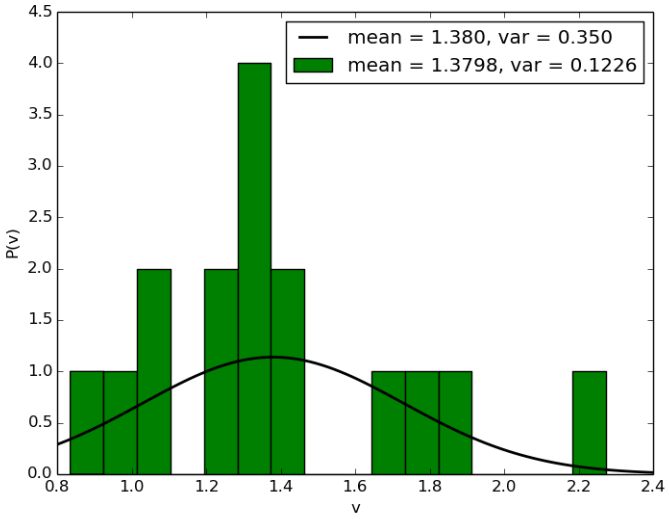

FIG. 1. The speed distribution of LJ patricles at equilibrium

PROBLEM 1

Simulate a system of 16 Lennard-Jones particles in a $2D$, periodic, 10×10 box(units $\epsilon = 1$, $\sigma = 1$, $\epsilon/k_B = 1$, reasonable value of the suggested time step in these units $\delta t = 0.01$, but explore whether that's appropriate).
a) Assuming that particles velocity components are distributed according to the Maxwell-Bolzmann distribution, find out what is the distribution of the speed $v = |\mathbf{v}|$. After some equilibration time, compare the observed speed distribution from the simulation with your derived expression.
b) Evaluate (t) and find the diffusion constant.
c) Is this a liquid, a solid, or a gas?
Technical: Use L-J potential truncated at $r = 3$; initial velocities homogeneously distributed in $(-3/2, 3/2)$, (ie.$-3/2 \leq v_x \leq 3/2$,etc), use the version of Verlet algorithm that is more accurate for the energy conversation.

Solution

We simulate the motion of 16 particles subject to the Lennard-Jones potential in a 10×10 box by applying the periodic boundary conditions. The motion of the particles in real time demonstrated in the file attached to the site as "moldynamics.mp4". The particles placed randomly around the lattice points have random initial velocities with the magnitude of 1.0.

a) After some equilibration time the average speed of the particles reach the distribution shown in the Fig.1 which resembles the Maxwell distribution which is given as

$$P(v) = C\frac{v^2}{k_B T} e^{-\frac{mv^2}{2k_B T}} \tag{1}$$

The magnitude of the initial speed was $\sqrt{2} \approx 1.4$. We can see from the figure that the mean of the distribution is 1.38 which is close to the initial speed.
b) We also the evaluate the $< |\mathbf{r}|^2 >$ of a single particle as a function of time steps in the FIG.2. Theoretically, for d dimensions it is given by

$$< |\mathbf{r}|^2 >= 2dDt, \tag{2}$$

where $D = 0.5$ is the diffusion coefficient. From our fit we obtained $D \approx 0.56$ which is pretty close to the true value.
c) As we saw in the previous section that the particles diffuse around inside the box which implies that our system resembles a gas. Also, the movie demonstrates this more vividly.

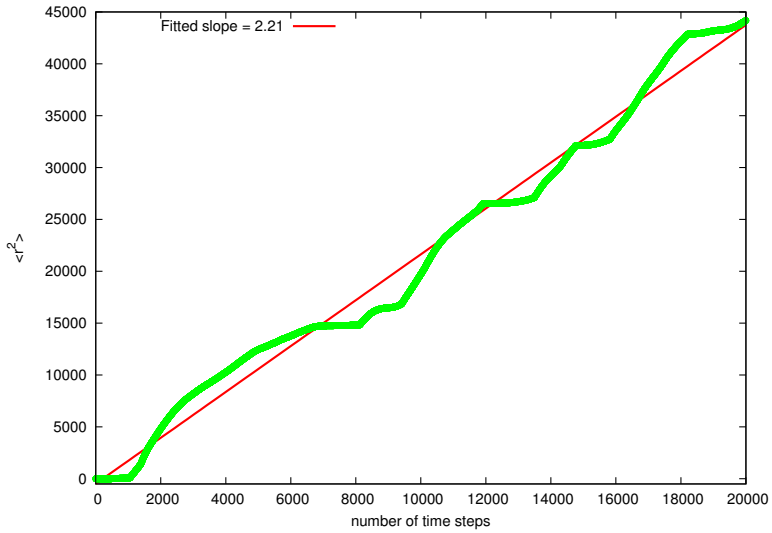

FIG. 2. The average of the square of the displacement after 20000 time steps

PROBLEM 2

Position 16 L-J particles on a perfect square lattice with the lattice constant $a_l = 1$ in a periodic box of the size 4×4. Assign random but tiny initial velocities to the particles, eg, 0.0001 or so. Since the particles are not at the equilibrium positions, the system should evolve. Use the time step 0.005 or smaller.
a)What is your guess for the value of temperature(after some equilibration time and using the reduced units as defined above)?
b)What is the structure of the equilibrium lattice which you see at $t > 15$ or so ?
c)How would you argue that the system models a $2D$ solid?

Solution

a) The temperature is estimated as from the following formula in the reduced units.

$$T = \sum_i \frac{v_i^2}{2(N-1)} \tag{3}$$

After some thermalization time the temperature of the system reached a value fluctuating around a constant value of 0.75 in reduced units as it is seen from the FIG.3.

b) After thermalization the system takes a form of a crystalline solid with a triangular lattice where the atoms vibrating around the equilibrium positions as the system has the minimum free energy in this form. It is depicted in the FIG.4.

c) The atoms in a $2D$ or any solid only vibrate around their equilibrium positions, do not wander around. Therefore, if our system is a $2D$ solid, the coordinates of a particle should just jiggle around the point, which is seen from the FIG.5.

PROBLEM3

Start from the resulting positions of problem 2. Gradually heat the system, ie, over a time interval of $\Delta t = 1$ increase the average temperature by small amounts ≈ 0.1 (in the units defined above).
a)Guess the melting temperature.(Consider supporting evidence in the form of graphics or displacements or pair correlations.)

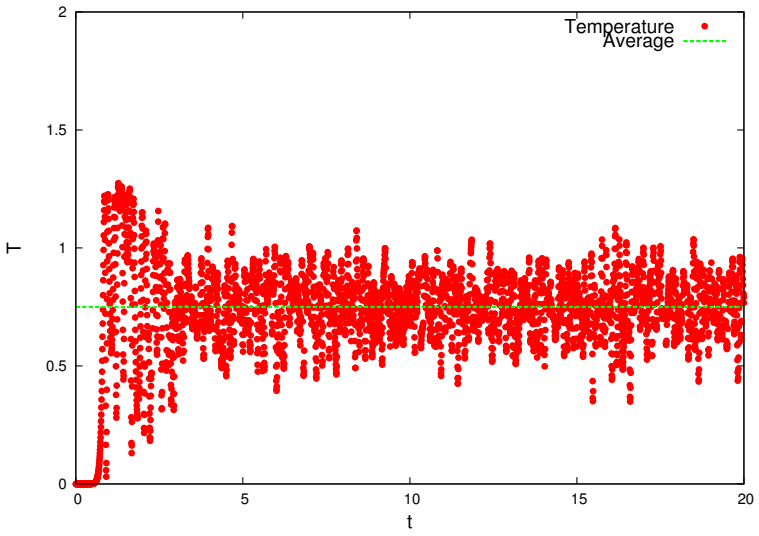

FIG. 3. The temperature of the system as a function of time

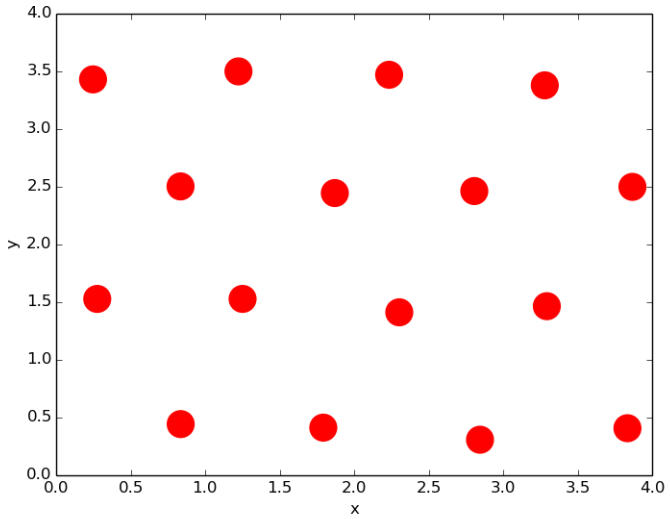

FIG. 4. The triangular lattice

Solution

a)

---

[1] N. J. Giordano and H. Nakanishi, **Computational Physics** (2nd edition, Prentice Hall, New Jersey, 2005).

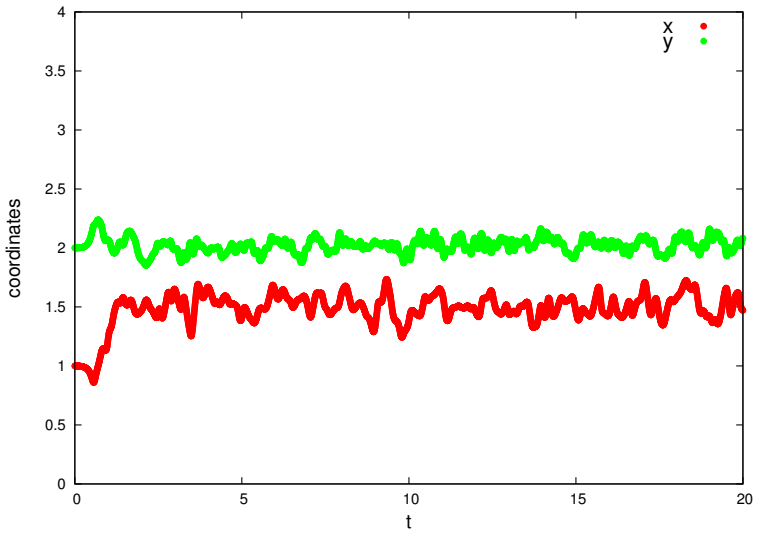

FIG. 5. The coordinates of a particle in the lattice as a function of time

APPENDIX: CODES

Molecular Dynamics

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
#include<vector>
#include<ctime>
#define random (double)rand()/(RAND_MAX +1.0)

  using namespace std;

struct Particle
{
  double xinit, yinit, xprev, yprev, xcur, ycur, xnew, ynew, vx, vy;
};

const int N = 4; //The number of particle in a slice
double energy(vector<Particle>, double);
double temp(vector<Particle>);
void r2(vector<Particle>,double &);
void heatup(vector<Particle> &);
void initialize(vector<Particle> &, double, double);
void initialize_lattice(vector<Particle> &, double, double);
void move(vector<Particle> &, double, double);
void relabel(vector<Particle> &);
main(){
  srand(time(NULL));
  double L = 4.0; //Size of the Box
  double dt = 0.005; //Time step
  double itmax = 200000;
  //Create N number of particles
  vector<Particle> particle(N*N);

  //Initialize the positions and the velocities of particles
  //initialize(particle, dt, L);
  initialize_lattice(particle, dt, L);
  double vx[N*N] = {0};
  double vy[N*N] = {0};
  double v[N*N]  = {0};
```

```
39    double r2mag = 0.0;
      //Evolve the system
41    for(int it = 0; it < itmax; it++ ){
        //Move the particles one step
43      move(particle, dt, L);

45      //Relabel the current position as previous
        relabel(particle);
47
        //Printout the physical quantities
49      int n = it%100;
        if(n==0){
51        char filename[256];
          sprintf(filename, "position/%03d.dat", it/100);
53        ofstream outfile(filename);
          for(int i = 0; i < N*N; i++){
55          outfile << particle[i].xcur << " " << particle[i].ycur << endl;
          }
57      }
        int m = it%4000;
59      if(m==0){
          //Heat up the system
61        heatup(particle);
        }
63      r2(particle, r2mag);
        cout << it*dt <<"  " << energy(particle, L) << " " << temp(particle) << " " << r2mag << " " <<
        particle[6].xcur << " "<< particle[6].ycur<<endl;
65      if(it>itmax*1/5){
          for(int i = 0; i < N*N; i++){
67          vx[i] += particle[i].vx;
            vy[i] += particle[i].vy;
69          v[i]  += sqrt(particle[i].vx*particle[i].vx + particle[i].vy*particle[i].vy );
          }
71      }
    }
73  ofstream outfile("velocity.dat");
    for(int i = 0; i < N*N; i++){
75    outfile << vx[i]/(itmax/4*5)<< "  " << v[i]/(itmax/4*5) << endl;
    }
77


79  }

81  void initialize(vector<Particle> &particle, double dt, double L){
      double v0 = 1.0;  //Initial velocity amplitude
83    double del = 0.5;
      int m, n;
85    for(int i = 0; i < N*N; i++){
          m = int(i/N);
87        n = int(i%N);
          particle[i].xcur = 2.0*(m+1) + 2*(random-0.5)*del;
89        particle[i].ycur = 2.0*(n+1) + 2*(random-0.5)*del;
          particle[i].xinit = particle[i].xcur;
91        particle[i].yinit = particle[i].ycur;
          particle[i].vx = 2*(random-0.5)*v0;
93        particle[i].vy = 2*(random-0.5)*v0;
          particle[i].xprev = particle[i].xcur - particle[i].vx*dt;
95        particle[i].yprev = particle[i].ycur - particle[i].vy*dt;
    }
97  }
    void initialize_lattice(vector<Particle> &particle, double dt, double L){
99    double v0 = 0.0001;  //Initial velocity amplitude
      int m, n;
101   for(int i = 0; i < N*N; i++){
          m = int(i/N);
103       n = int(i%N);
          particle[i].xcur = m;
105       particle[i].ycur = n;
          particle[i].xinit = particle[i].xcur;
107       particle[i].yinit = particle[i].ycur;
```

```
        particle[i].vx = pow(-1,rand()%2)*v0;
        particle[i].vy = pow(-1,rand()%2)*v0;
        particle[i].xprev = particle[i].xcur - particle[i].vx*dt;
        particle[i].yprev = particle[i].ycur - particle[i].vy*dt;
    }

}
void move(vector<Particle> &particle, double dt, double L){
    double fx[N*N];
    double fy[N*N];
    for(int k = 0; k < N*N; k++){
        for(int i = 0; i < N*N; i++){
            double r, rx, ry;
            double Fx = 0.0;
            double Fy = 0.0;
            for(int j = 0; j < N*N; j++){
                if(j==i) continue;
                rx = particle[i].xcur - particle[j].xcur;
                ry = particle[i].ycur - particle[j].ycur;
                if(abs(rx) > 0.5*L) rx *= -L/abs(rx) + 1.0;
                if(abs(ry) > 0.5*L) ry *= -L/abs(ry) + 1.0;
                r = sqrt(rx*rx + ry*ry);
                if(r > 3.0) continue;
                //if(r < 0.1) continue;
                Fx += 24*(2.0/pow(r,13) -1.0/pow(r,7))*rx/r;
                Fy += 24*(2.0/pow(r,13) -1.0/pow(r,7))*ry/r;
            }
            fx[i] = Fx;
            fy[i] = Fy;
        }
        particle[k].xnew = (2*particle[k].xcur - particle[k].xprev + fx[k]*dt*dt);
        if(particle[k].xnew > L) particle[k].xnew -= L;
        if(particle[k].xnew < 0) particle[k].xnew += L;
        particle[k].ynew = (2*particle[k].ycur - particle[k].yprev + fy[k]*dt*dt);
        if(particle[k].ynew > L) particle[k].ynew -= L;
        if(particle[k].ynew < 0) particle[k].ynew += L;
        particle[k].vx = (particle[k].xnew - particle[k].xprev)/(2*dt);
        particle[k].vy = (particle[k].ynew - particle[k].yprev)/(2*dt);
    }
}
void relabel(vector<Particle> &particle){
    for(int i = 0; i < N*N; i++){
        particle[i].xprev = particle[i].xcur;
        particle[i].yprev = particle[i].ycur;
        particle[i].xcur = particle[i].xnew;
        particle[i].ycur = particle[i].ynew;
    }
}
double energy(vector<Particle> particle, double L){
    double KE = 0.0;
    double PE = 0.0;
    double rx, ry, r;
    for(int i = 0; i < N*N; i++){
        for(int j = 0; j < N*N; j++){
            if(i==j) continue;
            rx = particle[i].xcur - particle[j].xcur;
            ry = particle[i].ycur - particle[j].ycur;
            if(abs(rx) > 0.5*L) rx *= -L/abs(rx) + 1.0;
            if(abs(ry) > 0.5*L) ry *= -L/abs(ry) + 1.0;
              r = sqrt(rx*rx + ry*ry);
            if(r > 3.0) continue;
            //if(r < 0.1) continue;
            PE += 2*(1.0/pow(r,12) -1.0/pow(r,6));
        }
        KE += 0.5*(particle[i].vx*particle[i].vx + particle[i].vy*particle[i].vy);
    }
    return KE + PE;
}
double temp(vector<Particle> particle){
    double KE = 0.0;
```

```
  for(int i = 0; i < N*N; i++){
    KE += 0.5*(particle[i].vx*particle[i].vx + particle[i].vy*particle[i].vy);
  }
  return KE/(N*N-1);
}

void heatup(vector<Particle> &particle){
  for(int i = 0; i < N*N; i++){
    particle[i].vx *= 1.2;
    particle[i].vy *= 1.2;
  }
}

void r2(vector<Particle> particle, double &r2mag){
  double delx, dely;
  int i = 5;
  //for(int i = 0; i < N*N; i++){
    delx = (particle[i].xcur - particle[i].xinit);
    dely = (particle[i].ycur - particle[i].yinit);
    r2mag += (delx*delx + dely*dely);
  //}
}
```

/home/omadillo/comp_physics/hw3/moldynamics.cc

Movie

```
from numpy import*
import matplotlib.pyplot as plt
import os, sys
os.system("rm md.mp4")
n = 0
while(n<100):
  x, y = loadtxt("position/%03d.dat"%n, unpack=True)
  plt.figure(n)
  plt.scatter(x,y,s=400,color='red')
  plt.xlabel('x')
  plt.ylabel('y')
  plt.xlim(0,4)
  plt.ylim(0,4)
  plt.savefig("position/%03d.png"%n)
  plt.close(n)
  n += 1
os.system(" cd position; ffmpeg -framerate 5 -pattern_type glob -i '*.png' -c:v libx264 ../md.mp4;
    cd ..")
```

/home/omadillo/comp_physics/hw3/plot.py

Speed

```
from numpy import*
import matplotlib.pyplot as plt
from scipy.stats import norm

vx, v = loadtxt("velocity.dat", unpack = True, usecols = [0,1])

plt.figure(1)
plt.hist(vx, bins = 16, color = 'green', label = 'mean = %.4f, var = %.4f'%(mean(vx),var(vx)));
mu,sigma = norm.fit(vx)
xmin,xmax = plt.xlim()
x = linspace(xmin,xmax,100)
p = norm.pdf(x,mu,sigma)
plt.plot(x,p,'k',linewidth=2,label='mean = %.3f, var = %.3f' %(mu,sigma))
plt.xlabel('vx')
```

```
plt.ylabel('P(vx)')
plt.legend()
plt.savefig("vx.png")

plt.figure(2)
plt.hist(v, bins = 16, color = 'green', label = 'mean = %.4f, var = %.4f'%(mean(v),var(v)));
mu,sigma = norm.fit(v)
xmin,xmax = plt.xlim()
x = linspace(xmin,xmax,100)
p = norm.pdf(x,mu,sigma)
plt.plot(x,p,'k',linewidth=2,label='mean = %.3f, var = %.3f' %(mu,sigma))
plt.xlabel('v')
plt.ylabel('P(v)')
plt.legend()
plt.xlim(0.8,2.4)
plt.ylim(0,4.5)
plt.savefig("v.png")
```

/home/omadillo/comp_physics/hw3/velocity.py