Chapter 6: Molecular Dynamics

The Lennard-Jones Solid and Liquid

I. Introduction

One of the principle uses of the molecular dynamics (MD) method is to study the thermodynamic and structural properties of solids and liquids. It has been used to study simple noble-gas systems, ionic materials, metals, covalent materials etc., employing the interatomic potentials that we have discussed in class. The purpose of this assignment is to introduce you to the methods used in MD and to give you some experience with its application.

Probably the most-studied material with atomistic simulations is "Lennard-Jonesium," i.e., a material described with a Lennard-Jones (LJ) interatomic potential. Here we show how one can perform molecular dynamics simulations on system of atoms described by that potential. We note that it would be easy to modify the following to use any pair potential (exp-6, Morse) and only somewhat more difficult to use more complicated potentials like the Embedded-Atom Model or a bond-order potential. We start by reviewing the LJ potential.

I. The Lennard-Jones Potential

The Lennard-Jones potential can be written in as

$$\phi_{LJ}(r) = 4\varepsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) \qquad (1)$$

We introduce reduced units as discussed in the text and in class: energy $E^* = E/\varepsilon$, potential energy $U^* = U/\varepsilon$, distance $r^* = r/\sigma$, temperature $T^* = k_B T/\varepsilon$, where k_B is the Boltzmann constant. The volume becomes $V^* = V/\sigma^3$. While the kinetic energy has units of energy, it is proportional to mv^2, which has units mass (distance/time)^2. To make it all consistent, we define a reduced time as $t^* = t/t_o$, where $t_o = \sqrt{m\sigma^2/\varepsilon}$.

The potential in reduced units is

$$\phi_{LJ}(r) = 4\left(\left(\frac{1}{r}\right)^{12} - \left(\frac{1}{r}\right)^{6}\right) \qquad (2)$$

where we have dropped the * and will just remember from now on that we are using reduced units.

As discussed in class, using cutoffs introduce discontinuities in the potential and force at the cutoff distance r_c . We could easily eliminate the discontinuity if the potential by *shifting* the energy by the value of the true potential at the cutoff, i.e., so that it goes to zero at the cutoff distance,

$$\phi_s(r) = \phi_{LJ}(r) - \phi_{LJ}(r_c) \tag{3}$$

We could then correct the energy at the end of the calculation. This does not eliminate the discontinuity in the force, however, which can lead to problems with energy convergence. For this exercise, *we will ignore both discontinuities*. However, if we were doing serious calculations for publication, we would want to introduce an interpolating function in the force and potential so that they go smoothly to 0 at r_c.

II. Forces and Pressure

For a central force potential like the Lennard-Jones form, the force on particle i is given by

$$\vec{F}_i = \sum_{j \neq i} \vec{f}_{ij} = \sum_{j \neq i} \left(-\frac{1}{r_{ij}} \frac{d\varphi}{dr_i} \right) \vec{r}_{ij} = \sum_{j \neq i} \frac{24}{r_{ij}^2} \left(2\left(\frac{1}{r_{ij}}\right)^{12} - \left(\frac{1}{r_{ij}}\right)^{6} \right) \vec{r}_{ij} \quad , \tag{4}$$

where the sum is over all the neighbors if *i* within a cutoff distance. We note again that this expression is in reduced units and we have dropped the *.

The pressure can be calculated by the equation given in the text and is (for a central force potential)

$$P = \frac{N}{V} k_B T + \frac{1}{3V} \left\langle \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{24}{r_{ij}^2} \left(2\left(\frac{1}{r_{ij}}\right)^{12} - \left(\frac{1}{r_{ij}}\right)^{6} \right) \right\rangle \tag{5}$$

where the angle brackets $\langle\ \rangle$ mean a thermodynamic average.

III. Solving the equations of motion: The Verlet algorithm

We use a very simple algorithm for solving the equations of motion - the Verlet algorithm. As discussed in the text, many other choices are possible. The Verlet algorithm has the advantage of being very simple to implement and is a generally satisfactory method.

As discussed in the class notes and the text, the equation to advance the position of the i^{th} atom from time *t* to $t+\delta t$ is

$$\vec{r}_i(t+\delta t) = 2\vec{r}_i(t) - \vec{r}_i(t-\delta t) + \vec{a}_i(t)\delta t^2 \qquad (6)$$

The accelerations $\vec{a}_i(t) = \vec{F}_i(t)/m_i$, where the forces are calculated as in Eq.(4) and m_i is the mass of particle *i.* The velocities are not calculated nor used directly in this method. We need their values, however, so that we can calculate the kinetic energy *K* (and thus temperature) and other quantities. We estimate their values at time *t* with the central difference formula

$$\vec{v}_i(t) = \frac{\vec{r}_i(t+\delta t) - \vec{r}_i(t-\delta t)}{2\delta t} \qquad (7)$$

The time step δt is chosen to ensure that the total energy (E=K+U) is well conserved, as discussed in the text.

From Eq.(6), we see that we need information from times *t* and *t-δt* to find the positions at time *t+δt.* The one time at which we do not have this information is at *t=0*. The initial positions $\vec{r}_i(0)$ and velocities $\vec{v}_i(0)$ are chosen in some way, as discussed below. We do not, however, have values for $\vec{r}_i(-\delta t)$ as needed in Eq.(6) to obtain $\vec{r}_i(+\delta t)$. We find $\vec{r}_i(-\delta t)$ with the equation

$$\vec{r}_i(-\delta t) = \vec{r}_i(0) - \vec{v}_i(0)\delta t + \frac{1}{2}\vec{a}_i(0)\delta t^2 \ , \qquad (8)$$

where the forces, and thus accelerations, are calculated using the initial positions, $\vec{r}_i(0)$.

IV. Initialization of positions and velocities

We need to initialize the atomic positions and the velocities. The atomic positions for the *N* atoms in the simulation will be chosen as needed for the problem of interest, as discussed below. For simulating condensed systems, we usually start from a solid structure, as described in the exercises for Chapter 3.

An issue that one faces when writing a molecular dynamics is whether one writes the positions in absolute or scaled coordinates (i.e., as fractions of the simulation cell size). So far, we have written all our lattice sum codes in terms of scaled coordinates, which I prefer as it makes it easy to change volumes, etc. Doing so, however, requires some care as we shall point out below.

The velocities can be chosen in a number of ways. Here we show how to choose the velocities from a Maxwell-Boltzmann distribution as discussed in the text (Section 6.9.2 and Appendix I.2.1). First, a value for the initial temperature is chosen: T_{init}. We then select each component of the velocity $\vec{v} = (v_x, v_y, v_z)$ for each atom from a distribution of the form

$$\rho(v_x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-v_x^2/2\sigma^2} \quad , \tag{9}$$

where the standard deviation is $\sigma = \sqrt{k_B T/m}$.

We can generate a set of random numbers from a Gaussian (normal) distribution with zero mean and unit standard deviation $\sigma = 1$ by choosing

$$\tau = \left(-\ln \Re_1\right)^{1/2} \cos\left(\pi \Re_2\right)$$

where $\Re_1$ and $\Re_2$ are real random numbers generated on (0,1). We start by assigning the velocities to this distribution. As discussed in the text, the net linear momentum is conserved. For our initial set of velocities, we calculate the average net linear momentum per atom. By subtracting that average from each velocity in the initial set, we ensure that the total linear momentum is zero at the beginning of the calculation. By doing so, we eliminate the possibility that the entire set of atoms drifts during the simulation (it make analysis a bit easier). Based on these velocities, we calculate the kinetic energy $K = \frac{1}{2}\sum_{i=1}^{N} m_i v_i^2$.

Since $K = 3Nk_BT/2$, the kinetic energy for the selected initial temperature is $K_{init} = 3Nk_BT_{init}/2$. By rescaling each velocity by the factor $\sqrt{K_{init}/K}$ we ensure that the kinetic energy from our input velocities equals that associated with the initial, prescribed, temperature.

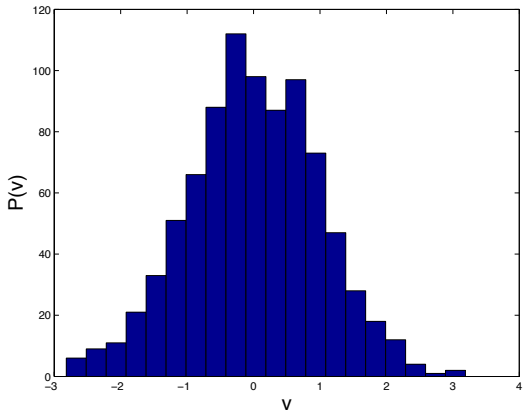
Note that these velocities refer to the time rate of change of the real coordinates of the atoms, not the scaled coordinates used in the simulation, which is correct. However, when we write the molecular dynamics code below, we will have the atom positions in the scaled coordinates. We will thus need to be a bit careful to make sure that we scale velocities and forces as appropriate in the code.

A code to create this distribution could look something like:

```
%  pick velocities from Maxwell-Boltzmann distribution
%  for any temperature we want.
%  Then we will calculate the kinetic energy and thus
%  the temperature of these atoms and then we will
%  rescale the velocities to the correct temperature
k = 0;
px = 0;
py = 0;
pz = 0;
for i=1:n
    vx(i) = sqrt(-2*log(rand))*cos(2*pi*rand);
```

```
    vy(i) = sqrt(-2*log(rand))*cos(2*pi*rand);
    vz(i) = sqrt(-2*log(rand))*cos(2*pi*rand);
    px = px + vx(i);
    py = py + vy(i);
    pz = pz + vz(i);
end
% find average momentum per atom
px = px/n;
py = py/n;
pz = pz/n;

set net momentum to zero and calcuate K
for i=1:n
    vx(i) = vx(i)-px;
    vy(i) = vy(i)-py;
    vz(i) = vz(i)-pz;
    k = k + vx(i)^2 + vy(i)^2 + vz(i)^2;
end
k = .5*k;
% kinetic energy of desired temperature (tin)
kin = 3*n*tin/2;
% rescale velocities
sc=sqrt(kin/k);
for i=1:n
    vx(i) = vx(i)*sc;
    vy(i) = vy(i)*sc;
    vz(i) = vz(i)*sc;
end
```



In this figure we plot the histogram of the velocities from a sample calculation. Note that aside from some fluctuations from being a finite sample, it is a normal distribution centered at 0.

V. Boundary conditions

For each molecular dynamics simulation, the boundary conditions must be prescribed. A typical calculation for a bulk solid material would be based on a unit cell with periodic boundary conditions. That cell could be cubic, for example for a cubic solid or a fluid, or it could be a more general symmetry based on one of the Bravia lattices, as discussed in the text and in the exercises for Chapter 3. Other types of calculations may employ different boundary conditions, for example free surfaces for examining surface structures.

For the bulk simulations described below, we will assume an approximation called the *minimum image convention*, as described in the text. Implementation of the minimum image convention is described in the exercise for Chapter 3. We again emphasize that

the minimum image convention was popular in the time of slow computers with small amounts of memory and small simulation cell sizes (small numbers of atoms). Since we will use small systems for these exercises, the minimum image approximation is appropriate. In research applications on fast computers with big systems, it is highly unlikely you would use this method, but would do a different type of lattice sum as described in the exercises for Chapter 3.

VI. Implementation: The force and energy calculations

As described in Eq.(6), at each time step we have a new set of atomic positions $\{\vec{r}\}$. To calculate the positions at the next time step requires the accelerations of each atom and, thus, the forces on that atom from the other atoms in the system, as described in Eq.(1). We also would calculate the potential energy, the kinetic energy, and likely the pressure and other quantities of interest.

As described in the exercises for Chapters 3 and 5, the potential energy at can be written as

$$U = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}{}' \phi_{ij}\left(r_{ij}\right) \quad . \tag{10}$$

We can also write the sums in Eq.(10) to avoid double counting the interactions,

$$U = \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} \phi_{ij}\left(r_{ij}\right). \tag{11}$$

Using Eq.(11) cuts the computation time in half.

To be computationally efficient, we will calculate *U* and the force with the same sums. To save time, we will use the version given in Eq.(11). In that sum, the interaction between each pair of atoms is only included once, which is fine for the potential energy. For the force, however, we need the the force exerted by atom *j* on atom *i*, $\vec{f}_{ij}$, as well as the force exerted by atom *i* on atom *j*, $\vec{f}_{ji} = -\vec{f}_{ij}$. We need to accumulate both terms as we sum over the interactions. How to do that is demonstrated in the following force code, in which we invoke the minimum image convention and a cutoff. Note that we also accumulate values needed to calculate the pressure, as shown in Eq.(5).

Note that the data structure is somewhat different in this code than in that presented in Chapter 3. In the earlier codes, we defined a vector `s(i,alpha)`, which returned the x components of `s` for alpha=1, y components for alpha=2, and z components for alpha=2. Here we use the variables `x(j), y(j), z(j)` for each atom. It makes the molecular dynamics code that we present below a bit easier to follow.

```
% simple lattice sum for force with cutoffs and
% minimum image convention
%
%  we calculate force (fx,fy,fz), energy (u), and
%  part of the pressure (w)
%
function[u,w,fx,fy,fz]= fLJsum(a,n,rc,x,y,z)
%  set force components, potential energy, and pressure to 0
%  fx,fy,fz are each vectors, with an entry for every atom
fx=zeros(n,1);
fy=zeros(n,1);
fz=zeros(n,1);
u = 0;
w = 0;
for i = 1:n-1  % note limits
    ftx = 0;
    fty = 0;
    ftz = 0;
    for j=i+1:n  %  note limits
% mimimum image convention
         xij = x(j)-x(i);
         yij = y(j)-y(i);
         zij = z(j)-z(i);
         xij = xij - round(xij);
         yij = yij - round(yij);
         zij = zij - round(zij);
         dist = a*sqrt(xij^2+yij^2+zij^2);

        if dist <= rc
            dphi = (2/dist^(12)-1/dist^6);
            ffx = dphi*a*xij/dist^2;
            ffy = dphi*a*yij/dist^2;
            ffz = dphi*a*zij/dist^2;
            ftx = ftx + ffx;
            fty = fty + ffy;
            ftz = ftz + ffz;
            phi = (1/dist^(12)-1/dist^6);
            u = u +  phi;
            w = w +  dphi;
%  add -f to sum of force on j
            fx(j) = fx(j) - ffx;
            fy(j) = fy(j) - ffy;
            fz(j) = fz(j) - ffz;
        end
    end
  %  sum up force on i (fi)
    fx(i) = fx(i) + ftx;
    fy(i) = fy(i) + fty;
    fz(i) = fz(i) + ftz;
```

```
end
%  need to multiply LJ by 4 and force and pressure by 24
%  also need to correct sign in f
u = 4*u;
w = 24*w;
for i=1:n
    fx(i) = -24*fx(i);
    fy(i) = -24*fy(i);
    fz(i) = -24*fz(i);
end
```

VII. Implementation: the Verlet algorithm

As noted above, we choose to write all the atomic positions in scaled coordinates. In the example code given next, we assume a cubic simulation cell with lattice parameter *a*. Please keep track of the places that *a* appears.

The parameters in this code are:

`nc`	number *fcc* unit cells used to generate the initial positions, as described in Chapter 3 exercies
`density`	density in reduced units
`tin`	input temperature in reduced units
`nsteps`	total number of time steps in the calculation
`dt`	time step in reduced units

The bare bones code is:

```
% MD code for 3D system of LJ atoms
%  input:  nc = number of cells of fcc unit cell
%          density = density of LJ system;
%          tin = initial temperature
%          nsteps = number of time steps
%          dt = time step

%  output:  un = potential energy at each time step
%           kn = kinetic energy at each time step
%           en = total energy at each time step
%           tn = temperature at each time step
%           pn = pressure at each time step
```

```
function[un,kn,en,tn,pn]= MDLJ(density,tin,nsteps,dt)

% initialize positions and velocities
[n,x,y,z,vx,vy,vz] = initLJMD(nc,tin);

% calculate some useful quantities
vol = n/density;
a = vol^(1/3);
rc = a/2;

% calculate initial energy and forces
% this calls the function fLJsum
[u,w,fx,fy,fz] = fLJsum(a,n,rc,s);

% now start the time stepping with the verlet algorithm
% initialize variables
xold = zeros(n,1);
yold = zeros(n,1);
zold = zeros(n,1);
xnew = zeros(n,1);
ynew = zeros(n,1);
znew = zeros(n,1);

% first find the positions at t-dt
for i=1:n
    xold(i) = x(i) - vx(i)*dt/a + .5*fx(i)*dt^2/a;
    yold(i) = y(i) - vy(i)*dt/a + .5*fy(i)*dt^2/a;
    zold(i) = z(i) - vz(i)*dt/a + .5*fz(i)*dt^2/a;
end

% start the time steps
for j=1:nsteps
    k = 0;
 %  find positions for time t + dt
 %  find velocities for time t
 %  find kinetic energy for time t
    for i=1:n
        xnew(i) = 2*x(i) - xold(i) + fx(i)*dt^2/a;
        ynew(i) = 2*y(i) - yold(i) + fy(i)*dt^2/a;
        znew(i) = 2*z(i) - zold(i) + fz(i)*dt^2/a;
        vx(i) = a*(xnew(i) - xold(i))/(2*dt);
        vy(i) = a*(ynew(i) - yold(i))/(2*dt);
        vz(i) = a*(znew(i) - zold(i))/(2*dt);
        k = k + vx(i)^2 + vy(i)^2 + vz(i)^2;
    end
    k = .5*k;
    temp = 2*k/(3*n);
%  create time series of values
```

```
    e = k + u;
    un(j) = u/n;
    kn(j) = k/n;
    en(j) = e/n;
    tn(j) = temp;
    pn(j) = density*temp + w/(3*vol);

% reset positions for next time step
    for i=1:n
        xold(i) = x(i);
        yold(i) = y(i);
        zold(i) = z(i);
        x(i) = xnew(i);
        y(i) = ynew(i);
        z(i) = znew(i);
    end
% calculate force and energy at new positions for next cycle
    [u,w,fx,fy,fz] = fLJsum(a,n,rc,x,y,z);
end
```

VIII. Determining whether the calculation is working

As noted in the text, we need to see if the calculation is working. There are two possibilities to consider: (1) Is the code written correctly? (2) Are we using it correctly? In molecular dynamics, we can answer both, to some degree, with one simple test. The total energy $E = K + U$ must be conserved, i.e., it should not change with time. Of course, numerical solutions to differential equations are never perfect, so calculated values of E will fluctuate over time. However, the value of E should fluctuate about an average value that shows no drift during the calculation and the range of the fluctuations should be small. A rule of thumb is that

$$\frac{(\max E - \min E)}{\langle E \rangle} \sim 10^{-4}, \tag{12}$$

where $(\max E - \min E)$ is the range of the values of E. Eq.(12) is just a guide that says that the range of the values of E should be much smaller than its value. Of course, that expression does not work well in the case where the average of E, $\langle E \rangle$, is small.

If the condition in Eq.(12) is not met, or if the values for E drift over time, then a few possibilities should be examined. (1) You may have an error in your code, which requires testing and debugging. Or, (2) you may have chosen your time step to be too big, which leads to inaccuracies in the solution to Newton's equations that could lead to drifting and large fluctuations in E. Another possibility is that you are dealing with potential functions that show large changes, which might mean that the method used to solve the equations of motion (in our case, the Verlet algorithm) is inadequate. In any case,

the minimum requirement for using the results from a molecular dynamics calculation is that the total energy is conserved.

Having energy conservations does not, however, guarantee a good calculation. You may have a perfectly good molecular dynamics code but have errors in your potential function. Those errors might be from miscoding, or you may have the wrong potential to model the system of interest. To check for these issues, one needs to compare to known simulations with the type of potential you are using (e.g., by repeating someone else's calculation) or to experiment. In the end, it is up to the simulator to both *verify* the code (i.e., to make sure it is correct) and *validate* the model (i.e., to make sure the model gives a good description of the system of interest).

VIII. Exercises:

There are two types of systems studied in these exercises. The first is a simple model of atoms on a surface, while the second is a study of the behavior of a bulk system described by an empirical potential.

Atoms on a surface:

This system is simpler to model and analyze than simulations of bulk systems. We will place atoms (starting with one) on a rigid surface with a simple form for the potential energy between the atom and the surface. There is energy exchange or coupled motions between the surface and the atoms moving on top of it. Furthermore, we take a simple form with parameters that are not designed to represent any real material. By varying those parameters, however, we can drive the system into many modes of behavior that are at least somewhat realistic.

We define the potential felt by the atom as it moves along the surface to be:

$$\phi(x,y) = A\sin(\pi x)\cos(\pi y) \ , \tag{12}$$

where *A* is an adjustable parameter that defines the strength of the interaction. The force is the negative of the gradient of the potential, as usual. We can define the following functions for the potential and force(with the constant given as acon):

```
% surface potential
function[phi]= phisurf(acon,x,y)
phi= acon.*sin(pi.*x).*sin(pi.*y);

% surface force
function[fx,fy]= fsurf(acon,x,y)
fx= -acon.*pi.*cos(pi.*x).*sin(pi.*y);
fy= -acon.*pi.*sin(pi.*x).*cos(pi.*y);
```

Let's look at these functions. First, note that we have used the multiplication form '.*' not '*', so that all members of a list would be included. This will make plotting easier. This

works for a single number (i.e., a list with length 1) as well. Note also that the force is a vector with two components.

We will describe how to develop a simulation to do one atom. Extension to more atoms will be part of the exercises.

If we are doing just one atom, it is difficult to define a "temperature" which is a thermodynamic quantity. Thus, we choose to use as an input parameter the total energy, which we will call `ein`. We will then pick a random position near a well, calculate the potential and use *K=E-U* to define the kinetic energy. Random velocities will be generated and then rescaled to give the proper *K.*

```
function[x,y,vx,vy]= initsurf(ein,acon)
%  pick x and y so they sit near a well
x=-.75+1.5*rand;
y=-.75+1.5*rand;
% calculate potential at x,y
phi=phisurf(acon,x,y);
%  find kinetic energy such that total energy is ein
ki= ein-phi;
% pick random velocity components
vxs = rand;
vys = rand;
% find kinetic energy from random velocities
ks = 0.5*(vxs^2+vys^2);
%  scale velocities so total energy is equal to ein
vx = vxs*sqrt(ki/ks);
vy = vys*sqrt(ki/ks);
```

The equations of motion will be solved using the Verlet algorithm as described above, remembering that we have only two dimensions in this case. To put the whole code together, we can link all the parts together, making sure that all the functions are called properly. In an attached folder, we present a completed code.

1. Using the Verlet algorithm, write and use an MD code to explore the “atom on surface” problem. Use E as an input parameter. Do not use periodic boundary conditions so that you can monitor large motions of your atoms. You can easily construct the code from the pieces given above.

   (a) Vary the time step to explore numerical error. Use plots of the calculated total energy as the measure of whether it is integrated properly.

   (b) Once you have determined an optimal time step, run calculations at a series of values of E, plotting the various energy terms to see the relation between potential, kinetic, and total energy.

   (c) Modify the code to calculate <u> and <K>. What is the temperature’s relation to <K>?

(d) For the same calculations, examine the behavior of the atom motion as a function of the energy of the atom. Explore the relationship between the input energy and the ability of the atom to jump out of the well.

(e) In all cases, plot the trajectory of the atom relative to the underlying potential. In MATLAB (and other packages), you can plot a contour of the potential as follows:

Define a vector of x values and a vector of y values, e.g.,

```
x = -3:.1:3;
y = 0:.5:10;
```

In this case, `x` has values -3,-2.9,-2.8,...,2.9,3 and `y` has values 0,.5,1,...9.5,10.

MATLAB has a function to convert those values into a 2 dimensional *grid* of values defined by the `x` coordinates (`X`) and `y` coordinates (`Y`)

```
[X,Y]=meshgrid(x,y);
```

If `f` is written as a function that uses the correct form for handling lists (i.e., with the '.*' type constructions shown in the coding for the force and potential given above), then we can write

```
Z = f(X,Y);
```

Now we have the data in the correct form for plotting. For example, `surf(X,Y,Z)` plots the surface, `mesh(X,Y,Z)` plots a grid, `contour(X,Y,Z)` makes a contour plot. Now suppose you want to *overlay* plots (for example, placing a line on a contour plot). You can use the `hold on` expression, for example

```
contour(X,Y,Z);
hold on
plot(xl,yl);
hold off
```

where `xl` and `yl` might contain the `x` and `y` values, respectively, of the beginning and end points of a line.

2. Modify the code to include more than one atom and with periodic boundary conditions using the minimum image convention. Choose a system size that includes at least 5 surface unit cells in each direction. Adding more than one atom will require adding the interactions between the atoms using the Lennard-Jones potential (or one of the other potentials from the exercises in Chapter 4). Above we show how to calculate the energy and forces. Note that as written above, the periodic surface is a square lattice with lattice length 1. The Lennard-Jones potential in reduced units has a minimum with $r_m = 2^{1/6} \approx 1.12$, so with no changes to the potential forms, the atoms will have a 12% mismatch at the surface. If you want to vary that mismatch, change the potential to be $\phi(x,y) = A\sin(\pi x / a)\cos(\pi y / a)$ where a is the lattice parameter of the surface atoms, which can be varied.

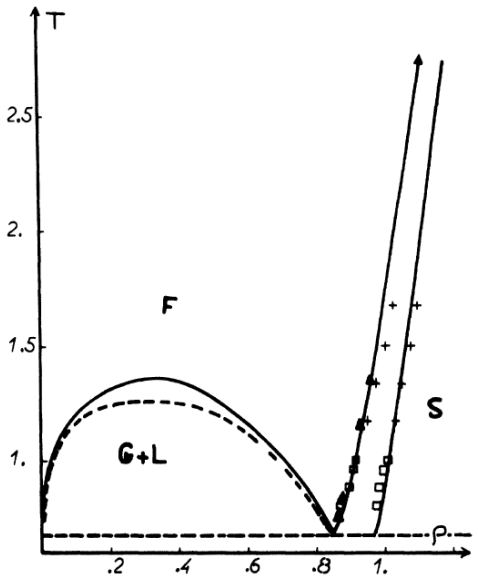
(a) Switch to a temperature-based choice for the initial velocities. Either use a normal distribution or pick a set of velocity components over a small range centered on zero and scale to the initial temperature as described above. Pick atomic positions near the minimum of the surface potential.

(b) Pick two temperatures and two values for *A*. Run calculations until the energy has equilibrated and then compare the structure of the atoms on the surface. Do they cluster together? Note that the atoms may move out of the initial simulation cell. While this will not be a problem for the energy calculation (the minimum-image approach takes care of that), it will cause problems with plotting the atom positions unless they are rescaled back in the central cell.

3. Repeat 2(b) for a different density of atoms on the surface.

Bulk material:

These calculations are for systems of atoms described by a Lennard-Jones potential in reduced units. The "phase diagram" of this material is well studied (of course only by computation). It is shown in the figure (from reference [1]).



You should restrict the densities (ρ) and temperatures (T) to stay in the solid (S) or liquid (L) regions. For reference, at *P=0*, the melting point is about $T = .7$.

4. We first need to establish the time step. We can often estimate it from what we know of the system we are studying. In the following, you will estimate the approximate time step for the simulation. Assume that reduced units, as described above, are used.

   (a) Use the mass of one of the rare gas atoms and the appropriate potential parameters from the text, and assume a time step of 5×10^{-15} sec. Work backward to find what that is in reduced units using the equations given above.

   (b) The period of an atom in a harmonic oscillator is $2\pi/\omega$, where $\omega = \sqrt{k/m}$. For an Lennard-Jones system at zero pressure and temperature, one can show that the force constant $k = 377\,\varepsilon/\sigma^2$ [2]. Assuming 50 time steps per period, calculate the time step in reduced units. Compare to the result in (a)

5. Assume some density and initial temperature for a LJ system. Use a small system, using a 2x2x2 set of *fcc* cells (N=32) or a 2x2x2 set of *fcc* cells (N=108). Assume the time step from problem 4.. Run the system for a few hundred steps. Using the figure of merit from Eq.(12), assess whether that time step is small enough. If not, lower it until the energy is conserved. If it is sufficient, then increase it until it is too large. The goal is to use the biggest time step possible that still gives a good solution. Note that for small systems, the inherent fluctuations are bigger so reaching 10^{-4} may not be possible.

5. Pick a density and temperature that would put the system into a *solid* phase. Using the optimal time step, run long enough (typically a few thousand time steps) to equilibrate the system and plot the various thermodynamic quantities (*E, T, K, P*). By what time step has the system equilibrated? Can you find average quantities over the equilibrated part of the run. It is sometimes easiest to output the calculated quantities as time sequences and then do all averaging separately from the main program. Note that in a long time sequence, do not include any values from times at which the system has not yet equilibrated.

6. You may have noticed that there can be a long equilibration time. The way the code is written it *always* starts from a perfect lattice and a set initial temperature. Thus, the system must equilibrate each time it is run, which can be very inconvenient. Another approach is to run the initialization routine separately (i.e., first), which sets an initial set of positions and velocities. Then one can change the first line of the MD code to read something like

```
function[un,kn,en,tn,pn,a,x,y,z,vx,vy,vz]= MDLJi(density,n-
steps,dt,n,x,y,z,vx,vy,vz)
```

and then remove the call to `initLJMD` in the MD code. Note that since the input changed in the MD code, I renamed it slightly.

Now the MD code reads in the positions and velocities, does a calculation, and outputs the *final* positions and velocities. A run could look like

```
[n,x,y,z,vx,vy,vz] = initLJMD(2,1);
[un,kn,en,tn,pn,a,x,y,z,vx,vy,vz]=MDLJi(.8,300,.005,n,x,y,z,vx,vy,vz);
```

After the MD calculation, x,y,z,vx,vy,vz are from the final configuration. So if we now did

```
[un,kn,en,tn,pn,a,x,y,z,vx,vy,vz]=MDLJi(3,.8,300,.005,n,x,y,z,vx,vy,vz);
```

we would start from where the previous calculation ended and would not have to wait for equilibration.

Modify the codes and redo the calculations from Problem 4, restarting the calculation until you have a stable, equilibrated system. Find averages. Analyze whether there

is drift in the calculation as discussed in class. Have you made a long enough run (i.e., have you used enough time steps) to get good values?

7. Pick a density and temperature that would put the system into a *liquid* phase. Using the optimal time step, run for a few thousand time steps (this should take a few minutes) and plot the various thermodynamic quantities. By what time step has the system equilibrated? Find average quantities over the equilibrated part of the run. Analyze whether there is drift in the calculation as discussed in class. Have you made a long enough run (i.e., have you used enough time steps) to get good values?

8. Using the time series of data that you already have, calculate fluctuations in the quantities, i.e., things like $\langle U \rangle^2 - \langle U^2 \rangle$. Note that these fluctuations do NOT give an estimate of the error in the calculation, but are related to thermodynamic quantities. To estimate error, we need to follow the procedure discussed in the text, binning the data and finding the standard deviation of the averages in the bins.

9. Compare solid and fluid calculations. You cannot really tell what state you are in from the energies. To know the phase, one needs to monitor the *structure* of the material. As discussed in the class and the text, an important quantity is the *radial distribution function g(r)*. It gives the probability that an atom is located a distance r away from another. It is normalized in such a way that an integral over *g(r)* over a distance gives the number of atoms in that distance. In the Appendix to the chapter on MD in the text, we discuss how to calculate *g(r)*. A simple code, which could be used to post process a set of atom positions, is below. It sums over all atom interactions and *bins* the distances (to make a histogram). It uses the normalization given the the appendix in the text.

```
%calculates g(r) for one set of atom positions
function[bp,ng]= gr(nbin,a,n,x,y,z)
rc = a/2;
xb = rc/nbin;
g = zeros(nbin,1);
bp = zeros(nbin,1);
ng = zeros(nbin,1);
%  bin all the distances
for i = 1:n-1 % note limits
    for j=i+1:n  %  note limits
        dx = x(j) - x(i);
        dy = y(j) - y(i);
        dz = z(j) - z(i);
        dx = dx - round(dx);
        dy = dy - round(dy);
        dz = dz - round(dz);
        dist = a*sqrt(dx^2 + dy^2 + dz^2);
        if  dist <= rc
            ib = floor(dist/xb);
            g(ib) = g(ib) + 1;
```

```
        end
    end
end
% normalize and create proper distances
factor = 2*a^3/(4*pi*n^2*xb);
for i=1:nbin
    bp(i) = (i+1/2)*xb;
    ng(i) = factor*g(i)/(i*xb)^2;
end
```

From the final configurations from Problems 6 and 7, calculate *g(r)* and plot it. (A command could look like `[r,g] = gr(20,a,n,x,y,z); plot(r,g);`) Can you tell the difference between the structures? One of the uncertainties you will face is that the data is noisy when you just use the final configuration. *g(r)* should be averaged over a long time sequence, as it also is an average quantity. One way to do this would be to modify the force routine (where you are already calculation distances) and then do the binning operation as shown above. Accumulate those bin values and then average at the end of the run, apply the factor in the same way and output. Try this.

10. In the text, a structural order parameter is introduced that discriminates between solids and liquids

$$\rho\left(\vec{k}\right)=\frac{1}{N}\sum_{i=1}^{N}\cos\left(\vec{k}\cdot\vec{r}_i\right)$$

where for an *fcc* lattice, a convenient choice is $\vec{k}=\left(2\pi/a_{fcc}\right)(-1,1,-1)$, where the a_{fcc} is the *fcc* unit cell size (not the simulation cell). Calculate this function for the final configuration of a run. Does is agree with expectations about whether the state is liquid or solid? Add to the MD code and track it through the calculation. How does its information compare with g(r)?

11. One of the most important quantities that one can calculate is the *velocity autocorrelation function*, defined as

$$c_{vv}(t)=\frac{m}{3k_BT}\left\langle\vec{v}(t)\cdot\vec{v}(0)\right\rangle$$

where the average is over all the particles. It is related to the diffusion constant, as discussed in class and in the text. Calculate this as a function of time for the solid and liquid. What differences do you see? Hint: You would need to make the following modifications to the code. Store the initial velocities, e.g., vxo, vyo, vzo. At each time step, calculate

```
cvv(k) = 0
```

```
for i=1:n
   cvv(k) = cvv(k) + vxo(i)*vx(i) + vyo(i)*vy(i) + vzo(i)*vz(i)
end
cvv(k) = cvv(k)/n
```

where `k` indicates the time step. Note that this is not a high-quality way to calculate the autocorrelation function. Better approaches are discussed in the text.

References

1. J. P. Hansen and L. Verlet, “Phase transitions of the Lennard-Jones system,” Phys. Rev. 184, 151–161 (1969).

2. R. LeSar and J. M. Rickman, “Finite-temperature properties of materials from analytical statistical mechanics,” Philosophical Magazine B 73, 627-639 (1996).