Chapter 3
Getting started with molecular dynamics modeling

Abstract In this chapter we provide a quick introduction to molecular dynamics modeling. In molecular dynamics the motion of a set of atoms is determined from a model for the inter-atom interactions. We demonstrate the basic physical formulation for a Lennard-Jones model for a gas and provide a Python implementation of the molecular dynamics algorithm. This Python implementation is too slow for any practical application, and we therefore introduce an open-source integrator to determine the motion of all the atoms. Installation and use of the LAMMPS simulator is described in detail. The simulation produces a set of trajectories for all the atoms in the model, and we also demonstrate how to read these trajectories into Python and use this data-set to characterize the behavior of realistic systems.

Statistical mechanics allows us to go from the atomic hypotesis to theories for macroscopic behavior. We will do this by developing strongly simplified models of the physical systems we are interested in: the ideal gas and the ideal crystal models. However, we can also study the behavior of a system in detail by calculating and following the motion of each atom in the system. In this book we will use a modeling tool called molecular dynamics. This is a useful tool that you may indeed use in your professional career as a researcher. Also, molecular dynamics allow you to pose and answer many questions about both atomic scale systems and many-particle systems — questions with non-trivial answers as well as real research questions. The tools we introduce here are indeed the tools that are used to understand nano-scale structures of materials, to understand and engineer chemical processes such as catalysis, to model and understand biological processes and to understand the interaction between drugs and living organisms. Molecular dynamics modeling is an important field of research and engineering, and our quick introduction here is in no way exhaustive, but should give you a quick introduction for the capabilities of the tool.

This chapter starts by motivating the use of classical mechanics to understand atomic motion. We introduce the Lennard-Jones model and standard boundary and initial conditions for atomic modeling studies. We introduce standard integration methods and a very simple illustrative implementation in Python. A more practical approach is then introduced using the LAMMPS software package, provide instal-

lation and simulation instructions for modeling the Lennard-Jones system using this package. We also demonstrate how to visualize and analyse the results of a simulation. Finally, we also demonstrate how LAMMPS can be used to model more realistic systems, such as water, biological and mineral systems. (These finals parts are interesting, but not central to exposition of this book, and can be skipped on a first reading).

3.1 Atomic modeling basics

How do we determine the motion of a system of atoms? This is really a problem in quantum mechanics — the motion and interaction of atoms, including the full motion of the electrons, needs to be determined from the basic equations of quantum mechanics. This is fully possible for small systems with present technology. However, to address the behavior of macroscopic systems, we need thousand, millions, or billions of atoms to get representative results from our calculations. However, if we want to model thousands to millions of atoms we need a different and more computationally efficient approach.

The use of the classical approximation to describe the motion of a system of atoms is, fortunately, able to capture many of the features of macroscopic systems. In this approximation, we use Newton's laws of motion and a well-chosen description of the forces between atoms to find the motion of the atoms. The forces are described by the potential energies of the atoms given their positions. The potential energy functions can, in principle, be determined from quantum mechanical calculations because the forces and the potential energies depend on the states of the electrons — it is the electrons that form the bonds between atoms and are the origin of the forces between atoms. However, in the classical approximation we parametrize this problem: We construct simplified models for the interactions that provide the most important features of the interactions.

3.1.1 Lennard-Jones potential

These parametrized models can be simple or complicated. They can include only pair interactions, three-particle interactions, or even many-particle interactions. Here, we will primarily use simple models, since the statistical and thermal effects we are interested in do not depend strongly on the details of the system. One of the simplest models for atom-atom interactions is a representation of the interactions between nobel-gas atoms, such as between two Argon atoms. For the interaction between two noble-gas atoms we have two main contributions:

- There is an attractive force due to a dipole-dipole interaction. The potential for this interactions is proportional to $(1/r)^6$, where r is the distance between the

atoms. This interaction is called the *van der Waals* interaction and we call the corresponding force the *van der Waals* force.
- There is a repulsive force which is a quantum mechanical effect due to the possibility of overlapping electron orbitals as the two atoms are pushed together. We use a power-law of the form $(1/r)^n$ to represent this interaction. It is common to choose $n = 12$, which gives a good approximation for the behavior of Argon.

The combined model is called the **Lennard-Jones potential**:

$$U(r) = 4\varepsilon \left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right) . \tag{3.1}$$

Here, ε is a characteristic energy, which is specific for the atoms we are modeling, and σ is a characteristic length.

The Lennard-Jones potential and the corresponding force $F(r)$ is illustrated in Fig. **??**. We see that the Lennard-Jones potential reaches its minimum when

$$F(r) = -\frac{d}{dr}U(r) = 0 , \tag{3.2}$$

which occurs at

$$F(r) = 24\,\varepsilon_0 \left((\sigma/r)^6 - 2\,(\sigma/r)^{12} \right) = 0 \;\Rightarrow\; \frac{r}{\sigma} = 2^{1/6} . \tag{3.3}$$

We will use this potential to model the behavior of an Argon system. However, the Lennard-Jones potential is often used not only as a model for a noble gas, but as a fundamental model that reproduces behavior that is representative for systems with many particles. Indeed, Lennard-Jones models are often used as base building blocks in many interatomic potentials, such as for the interaction between water molecules and methane and many other systems where the interactions between molecules or between molecules and atoms is simplified (coarse grained) into a single, simple potential. Using the Lennard-Jones model you can model 10^2 to 10^6 atoms on your laptop and we can model 10^{10}-10^{11} atoms on large supercomputers. However, if you are adventurous you may also model other systems, such as water, graphene, or complex biological systems using this or other potentials as we demonstrate in Sect. 3.5

3.1.2 Initial conditions

An atomic (molecular) dynamics simulation starts from an initial configuration of atoms and determines the trajectories of all the atoms. The initial condition for such

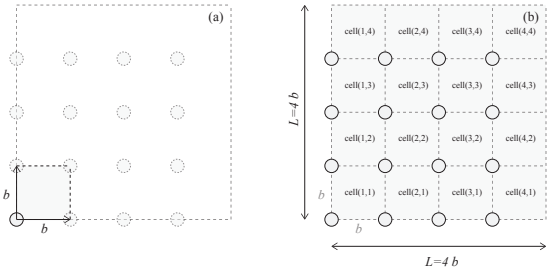

Fig. 3.1 **a** Illustration of a unit cell for a square lattice. **b** A system consisting of 4×4 unit cells, where each of the cells are marked and indexed for illustration.

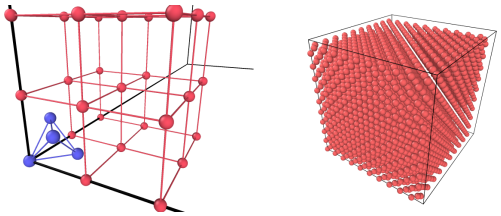

Fig. 3.2 *Left* Illustration of a unit cell for a face centered cubic lattice. Unit cell atoms illustrated in blue and the base position of other cells shown in red. *Right* A system consisting of $10 \times 10 \times 10$ unit cells.

a simulation consists of all the positions, $\mathbf{r}_i(t_0)$ and velocities $\mathbf{v}_i(t_0)$ at the initial time t_0. In order to model a realistic system, it is important to choose the initial configuration with some care. In particular, since most potentials such as the Lennard-Jones potential increase very rapidly as the interatomic distance r goes to zero, it is important not to place the atoms too close to each other. We therefore often place the atoms regularly in space, on a lattice, with initial random velocities.

We generate a lattice by first constructing a *unit cell* and then copying this unit cell to each position of a lattice to form a regular pattern of unit cells. (The unit cell may contain more than one atom). Here, we will use cubic unit cells. For a cubic unit cell of length b with only one atom in each unit cell, we can place the atom at $(0,0,0)$ in the unit cell and generate a cubic lattice with distances b between the atoms by using this cubic unit cell to build a lattice. This is illustrated for a two-dimensional system in Fig. 3.1. Such a lattice is called a *simple cubic lattice*.

However, for a Lennard-Jones system we know (from other theoretical, numerical and experimental studies) that the equilibrium crystal structure is not a simple cubic lattice, but a face centered cubic lattice. This is a cubic lattice, with additional atoms added on the center of each of the faces of the cubes. The unit cell for a face centered cubic lattice is illustrated in Fig. 3.2. We will use this as our basis for a simulation and then select a lattice size b so that we get a given density of atoms. The whole system will then consist of $L \times L \times L$ cells, each of size $b \times b \times b$ and with 4 atoms in each cell.

3.1.3 Boundary conditions

A typical molecular model of a liquid of Argon molecules is illustrated in Fig. 3.3a. In this case, we have illustrated a small system of approximately $10 \times 10 \times 10$ atom diameters in size. Below, you will learn how to set up and simulate such systems on your laptop. Unfortunately, you will not be able to model macroscopically large systems — neither on your laptop nor on the largest machine in the world. A liter of gas at room temperature typically contains about 10^{23} atoms, and this is simply beyond practical computational capabilities.

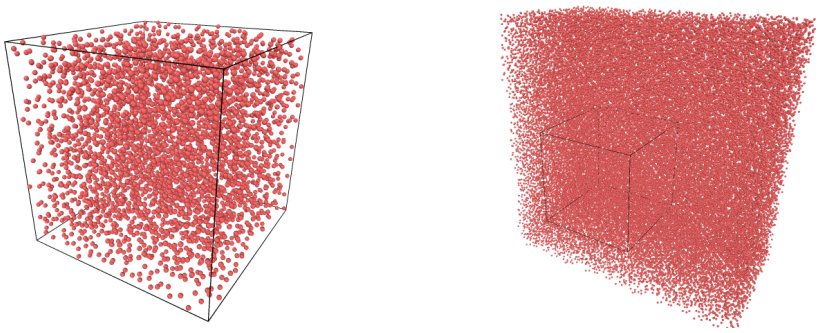

Fig. 3.3 **a** Visualization of a $10 \times 10 \times 10$ simulation of an Argon liquid. **b** Visualization of a 10×10 simulation of an Argon gas, showing the actual simulation area in the center box, and 8 copies of the center box illustrating the principles of the periodic boundary conditions.

But it is possible with a small system to gain some insights into how very large, even infinite, systems behave? One of the problems with the $10 \times 10 \times 10$ system above is the external boundaries. But we can fool the system into believing it is infinite by applying what we call periodic boundary conditions. If the atoms on the left of the system do not see emptiness to their left, but instead see the right hand side of the system, as if the system is wrapped around a cylinder, the system will look like it is inifinite. This is illustrated in Fig. 3.3b. This convention of periodic boundary conditions is usually applied in all simulations in order to avoid dealing with boundary conditions. (There may be some possibly problematic aspects of periodic boundaries, but we will not address or worry about these here).

3.1.4 Integrating the equations of motion

How do we determine the behavior of the system? We solve the equations of motion. For molecular dynamics simulations we usually use an algorithm called the Velocity-Verlet, which is approximately like the forward Euler method, but it is very well suited for conservative forces. The velocity is calculated at both time t and at

intermediate times $t + \Delta t/2$, where Δt is the time-step, whereas the forces are only calculated at the full time-steps, t, $t + \Delta t$, $t + 2\Delta t$ etc. The most time-consuming part of the calculation is the calculation of the forces. We therefore want to limit the number of times we calculate the forces and still have as high precision as possible in our calculations. The Velocity Verlet algorithm ensures a good trade-off between precision in the integration algorithm and the number of times forces are calculated.

In the Velocity-Verlet algorithm the positions $\mathbf{r}_i(t)$ and velocities $\mathbf{v}_i(t)$ of all the atoms, $i = 1, \ldots, N$, are propagated forward in time according to the algorithm:

$$\mathbf{v}_i(t + \Delta t/2) = \mathbf{v}_(t) + \mathbf{F}_i(t)/m_i \Delta t/2 \tag{3.4}$$

$$\mathbf{r}_i(t + \Delta t) = \mathbf{r}_(t) + \mathbf{v}_i(t + \Delta t/2) \tag{3.5}$$

$$\mathbf{F}_i(t + \Delta t) = -\nabla V(\mathbf{r}_i(t + \Delta t)) \tag{3.6}$$

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_(t + \Delta t/2) + \mathbf{F}_i(t + \Delta t)/m_i \Delta t/2\ ,, \tag{3.7}$$

This method has very good properties when it comes to energy conservation, and it does, of course, preserve momentum perfectly.

3.2 Simple implementation of a molecular dynamics simulator

How can we implement the full simulation procedure in Python? We need to set up the initial configuration of atoms, integrate the motion for a given number of time-steps, and then output the results to a file that can be read by standard visualization programs. Here, we provide a full implementation to show how a molecular dynamics simulation is implemented. However, this is mainly for illustrational purposes so that you can see how it is done. We will actually use a state-of-the-art open-source simulation package to perform our simulations.

3.2.1 Non-dimensional equations of motion

However, all the quantities in a molecular dynamics simulations are very small. It is therefore usual to introduce measurement units that are adapted to the task. For the Lennard-Jones model we usually use the intrinsic length and energy scale of the model as the basic units of length and energy. This means that we measure lengths in units of σ and energies in units of ε_0. A vector $\mathbf{r}'_i$ in the simulation is therefore related to the real-world length $\mathbf{r}_i$ through

$$\mathbf{r}_i = \sigma\, \mathbf{r}'_i \;\Leftrightarrow\; \mathbf{r}'_i = \mathbf{r}_i/\sigma\ . \tag{3.8}$$

Similarly, we can introduce a Lennard-Jones time, $\tau = \sigma\sqrt{m/\varepsilon}$, where m is the mass of the atoms, and the Lennard-Jones temperature $T_0 = \varepsilon/k_B$. Using these notations, we can rewrite the equations of motion for the Lennard-Jones system using the non-

dimensional position and time, $\mathbf{r}'_i = \mathbf{r}_i/\sigma$ and $t' = t/\tau$:

$$m\frac{d^2}{dt^2}\mathbf{r}_i = \sum_j 24\varepsilon_0 \left((\sigma/r_{ij})^6 - 2\,(\sigma/r_{ij})^{12} \right) \left(\mathbf{r}_{ij}/r_{ij}^2 \right) \ , \tag{3.9}$$

to become

$$\frac{d^2}{d(t')^2}\mathbf{r}'_i = \sum_j 24 \left(r_{ij}^{-6} - 2r_{ij}^{-12} \right) \left(\mathbf{r}'_{ij}/r'^2_{ij} \right) \ . \tag{3.10}$$

Notice that this equation is general. All the specifics of the system is now part of the characteristic length, time and energy scales σ, τ, and ε_0. For Argon $\sigma = 0.3405\mu$m, and $\varepsilon = 1.0318 \cdot 10^{-2}$eV, and for other atoms you need to find the corresponding parameter values.

Quantity	Equation	Conversion factor	Value for Argon
Length	$x' = x/L_0$	$L_0 = \sigma$	0.3405μm
Time	$t' = t/tau$	$\tau = \sigma\sqrt{m/\varepsilon}$	$2.1569 \cdot 10^3$ fs
Force	$F' = F/F_0$	$F_0 = m\sigma/\tau^2 = \varepsilon/\sigma$	$3.0303 \cdot 10^{-1}$
Energy	$E' = E/E_0$	$E_0 = \varepsilon$	$1.0318 \cdot 10^{-2}$ eV
Temperature	$T' = T/T_0$	$T_0 = \varepsilon/k_B$	119.74 K

3.2.2 Initialization of system

First, we need to initialize the system by generating a lattice of atoms with random velocities. We write a small script to generate a lattice and write it to disk. The lattice consists of units cells of size b (measure in units of σ, as are everything in the simulation). If a unit cell starts at $\mathbf{r}_0$, then the 4 atoms in the units cell are at positions $\mathbf{r}_0$, $\mathbf{r}_0 + (b/2, b/2, 0)$, $\mathbf{r}_0 + (b_2, 0, b_2)$, and $\mathbf{r}_0 + (0, b/2, b/2)$. If a system consists of $L \times L \times L$ such cells it is simple to create the system: We simply loop through all L^3 positions of $\mathbf{r}_0$ and for each such position we add atoms at the four positions relative to $\mathbf{r}_0$. This is done using the following script, which is explained in more detail below:

```
L = 5; % Lattice size
b = 2.0; % Size of unit cell (units of sigma)
v0 = 1.0; % Initial kinetic energy scale
N = 4*L^3; % Nr of atoms
r = zeros(N,3);
v = zeros(N,3);
bvec = [0 0 0; b/2 b/2 0; b/2 0 b/2; 0 b/2 b/2];
ip = 0;
% Generate positions
for ix = 0:L-1
    for iy = 0:L-1
```

```
        for iz = 0:L-1
            r0 = b*[ix iy iz]; % Unit cell base position
            for k = 1:4
                ip = ip + 1; % Add particle
                r(ip,:) = r0 + bvec(k,:);
            end
        end
    end
end
% Generate velocities
for i = 1:ip
    v(i,:) = v0*randn(1,3);
end
% Output to file
writelammps('mymdinit.lammpstrj',L*b,L*b,L*b,r,v);
```

In addition, we need the following function to write the data to file:

```
function writelammps(filename,Lx,Ly,Lz,r,v)
%WRITELAMMPS Write data to lammps file
fp = fopen(filename,'w');
s = size(r);
ip = s(1);
fprintf(fp,'ITEM: TIMESTEP\n');
fprintf(fp,'0\n');
fprintf(fp,'ITEM: NUMBER OF ATOMS\n');
fprintf(fp,'%d\n',ip); % Nr of atoms
fprintf(fp,'ITEM: BOX BOUNDS pp pp pp\n');
fprintf(fp,'%f %f\n',0.0,Lx); % box size, x
fprintf(fp,'%f %f\n',0.0,Ly); % box size, y
fprintf(fp,'%f %f\n',0.0,Lz); % box size, z
fprintf(fp,'ITEM: ATOMS id type x y z vx vy vz\n');
for i = 1:ip
    fprintf(fp,'%d %d %f %f %f %f %f %f \n',...
            i,1,r(i,:),v(i,:));
end
fclose(fp);
end
```

Notice that we use the vectors $\mathbf{b}_k$ to describe the four positions relative to the origin, $\mathbf{r}_0$ of each cell:

```
bvec = [0 0 0; b/2 b/2 0; b/2 0 b/2; 0 b/2 b/2];
...
r0 = b*[ix iy iz]; % Unit cell base position
for k = 1:4
    ip = ip + 1; % Add particle
    r(ip,:) = r0 + bvec(k,:);
end
```

Also notice that we use a variable `ip` to keep track of the particle index for the atom we are currently adding to the system. This is simpler than calculating the particle number each time. That is, we can simply use

```
    ip = ip + 1; % Add particle
```

instead of the more elaborate and less transparent

```
    ip = ix*4*L*L + iy*4*L + iz
```

to calculate the current particle number each time a new atom is added. Finally, we add a random velocity vector, with a normal (Gaussian) distribution of velocities with a standard deviation of v_0 using the `randm` function. This can be done using a loop or with a single, vectorized command:

```
for i = 1:ip
    v(i,:) = v0*randn(1,3);
end
#
```

or

```
v = v0*randn(ip,3);
```

Finally, the state is saved to the file `mymdinit.lammpstrj`. The resulting file, which contains the complete atomic state of the system, looks like this:

```
ITEM: TIMESTEP
0
ITEM: NUMBER OF ATOMS
500
ITEM: BOX BOUNDS pp pp pp
0.000000 10.000000
0.000000 10.000000
0.000000 10.000000
ITEM: ATOMS id type x y z vx vy vz
1 1 0.000000 0.000000 0.000000 -0.306633 -2.732455 1.612753
2 1 1.000000 1.000000 0.000000 0.099804 0.487968 -1.545902
3 1 1.000000 0.000000 1.000000 -0.500267 0.777696 0.028699
4 1 0.000000 1.000000 1.000000 -0.404407 -0.867741 0.626161
...
```

where we only have included the first four of 500 atoms. This file can then be used as a starting point for a simulation. The output from a simulation will have a similar format, providing the state of the atomic system which we can then analyze in details.

3.2.3 Integration of the equations of motion

Starting from the initial configuration, we integrate the equations of motion to find the particle trajectories and output them to file for further analysis. First, we write a short function to read the initial configuration from file, and then we integrate and write results to a file.

The function readlammps[1] used to read from a LAMMPS trajectory file will be reused many times throughout this book, hence we strive to make it sufficiently general and sophisticated. You can find the listing and discussion of this program in the Appendix (Chap. **??**). Here, we simply notice that we can call it as

```
[Lx,Ly,Lz,r,v]=readlammps('mymdinit.lammpstrj');
```

In order to read in the initial state. The following program integrates the equation of motion:

```
[Lx,Ly,Lz,r,v]=readlammps('mymdinit.lammpstrj');
L = [Lx Ly Lz]; s = size(r); N = s(1);
t = 3.0; dt = 0.001; n = ceil(t/dt);
a = zeros(N,3);  % Store calculated accelerations
for i = 1:n-1 % Loop over timesteps
    a(:,:) = 0;
    for i1 = 1:N
        for i2 = i1+1:N
            dr = r(i,i1,:) - r(i,i2,:);
            for k = 1:3  % Periodic boundary conditions
                if (dr(k)>L(k)/2) then
                    dr(k) = dr(k) - L(k);
                end
                if (dr(k)<-L(k)/2) then
                    dr(k) = dr(k) + L(k);
                end
            end
            rr = dot(dr,dr);
            aa = -24*(2*(1/rr)^6-(1/rr)^3)*dr/rr;
            a(i1,:) = a(i1,:) + aa(1); % from i2 on i1
            a(i2,:) = a(i2,:) - aa(2); % from i1 on i2
        end
    end
    v(i+1,:,:) = v(i,:,:) + a*dt;
    r(i+1,:,:) = r(i,:,:) + v(i+1,:,:)*dt;
    % Periodic boundary conditions
    for i1 = 1:N
        for k = 1:3
            if (r(i+1,i1,k)>L(k) then
                r(i+1,i1,k) = r(i+1,i1,k) - L(k);
            end
             if (r(i+1,i1,k)<0 then
                r(i+1,i1,k) = r(i+1,i1,k) + L(k);
            end
        end
    end
end
writelammps('mymddump.lammpstrj',Lx,Ly,Lz,r,v);
```

Notice how the periodic boundary conditions are implemented. They need to be included both when the relative distance $\Delta \mathbf{r}_{ij}$ between two atoms are calculated and when the positions of the atoms are updated during the integration step.

[1] http://folk.uio.no/malthe/fys2160/readlammps.m

The main idea here is to show you the structure and principles of a molecular dynamics program, as well as how data is handled and stored in files. This Python program is not practically useful because Python is too slow, in particular for a program using many nested loops. We will therefore not use this code, but instead use a professionally developed open-source molecular dynamics integrator to find the time development of the atomic system. The principles are the same, but the computational efficiency is many orders of magnitude greater for a professional code.

3.3 Running a molecular dynamics simulation

There are several efficient packages that solves the equations of motion for a molecular dynamics simulation. The packages allow us to model a wide variety of systems, atoms and molecules, and are efficienty implemented on various computing platforms, making use of modern hardware on your laptop or desktop or state-of-the-art supercomputing fascilities. We use a particular tool developed at Sandia National Laboratories called LAMMPS[2].

3.3.1 Installation of LAMMPS

If you want to be able to reproduce the simulations performed here you will need to install LAMMPS on your computer. This is very simple if you have a Mac or an Ubuntu system — for a windows system you will have to follow the installation instructions found at the web-site for LAMMPS. You can find all the documentation of LAMMS here[3].

Mac installation. On a mac you should be able to install LAMMS using Homebrew or Macports.

Using **Homebrew** (Homebrew[4]) LAMMPS is installed with:

```
brew tap homebrew/science
brew install lammps
```

If you want to use the parallell implementation of LAMMPS you can install this version using

```
brew tap homebrew/science
brew install lammps --with-mpi
```

[2] `http://lammps.sandia.gov/`

[3] `http://lammps.sandia.gov/`

[4] `http://brew.sh/`

Using **MacPorts** (MacPorts[5]) LAMMPS is installed with:

```
port install lammps
```

Ubuntu installation. You can install a recent version of LAMMPS with:

```
sudo apt-get install lammps
```

Python interface installation. In addition you should install the module `Pizza.py` which we use to read simulation data into Python, and you need a Python installation that includes `pylab`. I recommend Anaconda[6], but the Enthought version also works fine. Download Pizza.py[7] and follow the installation instructions.

3.3.2 Starting a simluation in LAMMPS

If you have successfully installed LAMMPS, you are ready to start your first molecular dynamics simulations. The LAMMPS simulator reads its instructions on how to run a simulation from an input file with a specific syntax. Here, we will set up a two-dimensional simulation of a Lennard-Jones system using the file `in.myfirstmd`:

```
# 2d Lennard-Jones gas
units lj
dimension 2
boundary p p p
atom_style atomic

lattice hex 0.75
region simbox block 0 20 0 10 -0.1 0.1
create_box 1 simbox
create_atoms 1 box

mass 1 1.0
velocity all create 2.5 87287

pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5

neighbor 0.3 bin
neigh_modify every 20 delay 0 check no

fix 1 all nve

dump 1 all custom 10 dump.lammpstrj id type x y z vx vy vz
thermo 100
run 5000
```

[5] `https://www.macports.org/`

[6] `http://continuum.io/downloads`

[7] `http://pizza.sandia.gov/`

The simulation is run from the command line in a Terminal. Notice that the file `in.myfirstmd` must be in your current directory. I suggest creating a new directory for each simulation, copying the file `in.myfirstmd` into the directory and modifying the file to set up your simulation, before starting the simulation with:

```
lammps < in.myfirstmd
```

This simulation should only take a few seconds. It produces output in the Terminal and two files: `dump.lammpstrj`, which contains all the data from the simulation, and `log.lammps`, which contains a copy of what was output to your terminal. Fig. 3.4 illustrates the positions of the atoms initially and at the end of the simulation.

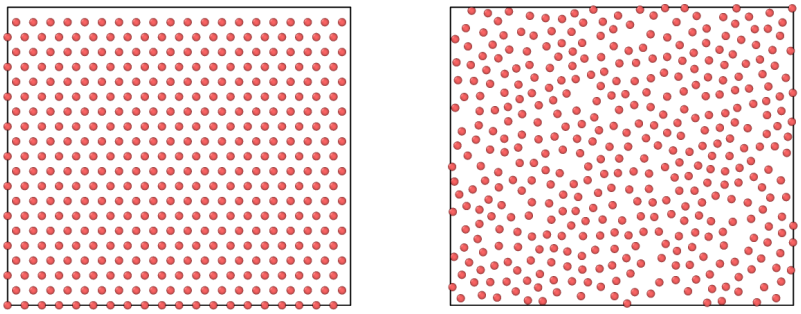

Fig. 3.4 (*Left*) Illustration of the initial hexagonal lattice generated in the simulation. (*Right*) Illustration of the final positions of the atoms after 5000 timesteps.

The input file `in.myfirstmd` consists of a series of commands to be interpreted by LAMMPS. Here, we look at what these do in detail. (You can skip this at first reading, and return when you wonder what the parameters actually do).

```
# 2d Lennard-Jones gas
```

This line starts with a # and is a comment that is ignored by the program.

```
units lj
dimension 2
boundary p p p
atom_style atomic
```

This block describes general features of the simulation:

The `units lj` command selects Lennard-Jones units, which were introduced in Sect. 3.2.1. This means that lengths are measured in units of σ, energies in units of ε_0, time in units of $\tau = \sigma\sqrt{m/\varepsilon}$, and temperature in terms of

$T_0 = \varepsilon/k_B$. For Argon $\sigma = 0.3405\mu$m, and $\varepsilon = 1.0318 \cdot 10^{-2}$eV. Other atomic models will have other parameters.

The `dimension` command specifies the dimensionality of the simulation: 2 or 3. Here we run a 2d simulation.

The `boundary` command specifies boundary conditions to be applied. Here we have periodic boundaries in the x-, y-, and z- directions.

The `atom_style` command specifies the complexity of the description of each atom/particle. Here, we will use the simplest description, `atomic`, which is used for noble gases and coarse-grained simulation models.

```
lattice hex 0.75
region simbox block 0 20 0 10 -0.1 0.1
create_box 1 simbox
create_atoms 1 box
```

This block sets up the dimensions of the 20×10 simulation box and fills the box with atoms with a given packing fraction.

The `lattice` command generates a lattice of points. This does, surprisingly enough, not actually generate any atoms, it only generates a set of positions in space where atoms will be generated when we generate atoms. The type `hex` specifies a two-dimensional lattice of hexagonal shape, so that each atom has six nearest neighbors. And the number `0.75` is called the scale and is the reduced density, ρ', when we have chosen LJ units for the simulation. (Notice that the scale is interpreted differently if we do not use LJ units, see the LAMMPS documentation for more information).

The `region` command defines a region which is a `block` extending over $0 < x < 20$, $0 < y < 10$, $-0.1 < z < 0.1$ We give this region the name `simbox`.

The `create_box` command now actually creates the simulation box based on the spatial region we called `simbox`. The simulation box will only contain 1 (one) type of atoms, hence the number `1`.

The `create_atoms` finally fills the simulation box we have defined using the lattice we have defined with atoms of type `1`.

```
mass 1 1.0
velocity all create 2.5 87287
```

This block defines the material properties of the atoms and defines their initial velocities.

The `mass` command defines that atoms of type `1` will have a mass of `1.0` relative to the mass of the Lennard-Jones model. This means that all atoms have mass 1 in the Lennard-Jones units. This means that the masses of all the atoms are the same as the mass m used in the non-dimensionalization of the Lennard-Jones model.

The `velocity` command generates random velocities (using a Gaussian distribution) so that the initial temperature for `all` atom types in the system is `2.5` in the dimensionless Lennard-Jones units. The last, strange integer number `87287` is the seed for the random number generator used to generate the

random numbers. As long as you do not change the seed number you will always generate same initial distribution of velocities for this simulation.

```
pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5
```

This block specifies the potential between the atoms.

The `pair_style` command specifies the we want to use a Lennard-Jones potential with a cut-off that is of length `2.5`. What does this mean? It means that since the Lennard-Jones potential falls so quickly to zero as the distance between the atoms increase, we will approximate interaction to be zero when the atoms are further than 2.5 away from each other (measured in Lennard-Jones units, that is in units of σ). The simulator ensures that both the potential and the force is continuous across the transition. There are many other types of force fields that you may use — take a look at the documentation of LAMMPS for ideas and examples.

The `pair_coeff` command specifies the parameters of the Lennard-Jones model. The two first numbers, `1 1`, specifies that we describe the interaction of atoms of type 1 with atoms of type 1. And the parameters of the Lennard-Jones model are `1.0 1.0 2.5`. This means that The interaction between an atom of type 1 with an atom of type 1 has a σ-value corresponding 1.0 times the the general σ-value (hence the first number 1.0), and a ε_0-value corresponding to 1.0 times the overall ε-value (hence the second number 1.0). The cut-off for this interaction is 2.5 — the same value as we specified above.

```
neighbor 0.3 bin
neigh_modify every 20 delay 0 check no
```

This block contains information about how the simulator is to calculate the interaction between the atoms using list of neighbors that are updated at regular intervals. You do not need to change these parameters, but changing them will typically not have any effects on the simulation results, only on the efficiency of the simulations.

```
fix 1 all nve
```

This one-line block specifies what type of simulation we are performing on the atoms. This is done by one or more `fix` commands that can be applied to regions of atoms. Here, the `fix`, which we call `1` (you can choose numbers or names for identity), is applied to `all` particles and specifies that the simulation is run at constant `nve`, that is, at constant number of particles (`n`), constant volume (`v`, meaning that the simulation box does not change during the simulation), and constant energy (`e`). You may be surprised by the constant energy part. Does the integration algorithm ensure that the energy is constant. Yes, it does. However, there can be cases where we want to add energy to a particular part of the system, and in that case the basic interation algorithm still conserves energy, but we add additional terms that may change the total energy of the system.

```
dump 1 all custom 10 dump.lammpstrj id type x y z vx vy vz
thermo 100
run 5000
```

This block specifies simulation control, inclusing output and the number of time-steps to simulate.

The `dump` command tells the simulator to output the state. The `1` is the name we give this dump — it could also have been given a name such as `mydump`. We specify that `all` atoms are to be output using a `custom` output format, with output every `10` time-steps to the file `dump.lammpstrj`, and the 'id type x y z vx vy vz' list specifies what is output per atom.

The `thermo` command specifies that output to the Terminal and to the log file, `log.lammps`, is provided every `100` timesteps.

The `run` command starts the simulation and specifies that it will run for `5000` timesteps.

3.3.3 Visualizing the results of a simulation

It is good practice to look at the results of the simulation. Use for example VMD[8] or Ovito[9] to visualize the results. Here, let us demonstrate the use of VMD. First, open VMD. Then open the `File -> New Molecule` menu item. You find the `dump.lammpstrj` file that were generated by your simulation run, and press load. The whole time sequence of your simulation is now loaded in VMD. However, the default visualization mode is usually not that clear. You fix that by going to the main menu again, and select `Graphics -> Representations...` In the window that opens you change the `Drawing method` to `CPK` and you select `Bond Radis` to be 0.0. Now, your visualization should show the time dynamics of an ensemble of atoms. Enjoy!

3.4 Analyzing the results of a simulation

The results from the simulations can be analyzed using built-in tools from LAMMPS. We demonstrate these tools by measuring the number of particle on the left side of the box as a function of time, by extracting that data from the simulation files.

We have illustrated the text-format output files from LAMMPS above. The file may contain data from may timesteps, t. For each timestep, there are a few initial lines describing the system, followed by a list of properties for each atom. We have chosen the custom output form

[8] `http://www.ks.uiuc.edu/Research/vmd/`

[9] `http://www.ovito.org/`

```
dump 1 all custom 10 dump.lammpstrj id type x y z vx vy vz
```

Each line of atom properties contains data of the form `id type x y z vx vy vz`. This means that we will get access to both the atom position and the atom velocity for each of the atoms if we read the data in this file. Our strategy will then be to use the atom position x_i and compare it with $L_x/2$, where L_x is the dimensions of the simulation box, measured using the same units as used for the positions x_i of the atoms. How can we read the data from the LAMMPS dump files. We could read our own input functions, but it is simpler to use already developed code, which is distributed with the LAMMPS distribution.

3.4.1 Matlab implementation

For matlab, we use the tools found in the `tools/matlab` directory, which was part of the lammps installation. (If you cannot find these after installing LAMMS, you can always download LAMMS again as a tarball, extract the tarball, and find the programs in the resulting directory). You need to copy the file `readdump_all.m` and `readdump_one.m` to your current working directory — the directory where you ran the simulation and where the file `dump.lammpstrj` is located.

First, we read data from all the timesteps into matlab:

```
data = readdump_all('dump.lammpstrj');
```

We need the time values t_i at each of the timesteps and the number of time-steps, `nt`:

```
t = data.timestep;
nt = length(t);
```

We set up an array, `nleft`, to store $n(t)$, the number of atoms on the left side as a function of time:

```
nleft = zeros(nt,1);
```

The size of the simulation box is stored in the variable `box`, and we store $L_x/2$ in the variable `halfsize`:

```
box = data.x_bound;
halfsize = 0.5*box(:,2);
```

Now, we loop through all the timesteps. For each timestep we extract all the x-positions for all the atoms in a list `xit`. Next, we find a list of all the atoms that are on the left side, that is, all the atoms with an x-position smaller than $L_x/2$. This is done by the `find` command. Finally, we count how many elements are in this list. This is the number of atoms that are on the left side of the box. We store this number in the array `nleft` for this timestep, and reloop to address the next timestep:

```
for it = 1:nt
```

```
    xit = data.atom_data(:,3,it);
    jj = find(xit<halfsize(it));
    nleft(it) = length(jj);
end
```

The complete program reads:

```
%start1
data = readdump_all('dump.lammpstrj');
t = data.timestep;
nt = length(t);
nleft = zeros(nt,1);
box = data.x_bound;
halfsize = 0.5*box(:,2);
for it = 1:nt
    xit = data.atom_data(:,3,it);
    jj = find(xit<halfsize(it));
    nleft(it) = length(jj);
end
plot(t,nleft), xlabel('t'), ylabel('n')
```

A short and simple program to handle a complicated set of data. This should provide you with a good basis for measuring various properties of the simulated system using Python.

3.4.2 Python implementation

For Python we use the tool `Pizza.py`. You should then simply start your python session by calling `pizza.py` initially. If we assume that you have installed the program in `/pizza`, you can start your `ipython session` using `pizza.py` by typing

```
ipython -i ~/pizza/src/pizza.py ~/pizza/src/pizza.py
```

in your terminal. The you are ready to analyze data from the simulation.

First, we read data from all the timesteps into Python:

```
from pylab import *
data = dump("dump.lammpstrj") # Read all timesteps
```

We need the time values t_i at each of the timesteps and the number of time-steps, `nt`:

```
t = data.time()
nt = size(t)
```

We set up an array, `nleft`, to store $n(t)$, the number of atoms on the left side as a function of time:

```
nleft = zeros(n,float); # Store number of particles
```

The size of the simulation box is found in the variable `box`, and we store $L_x/2$ in the variable `halfsize`:

```
tmp_time,box,atoms,bonds,tris,lines = data.viz(0)
halfsize = 0.5*box[3] # Box size in x-dir
```

Now, we loop through all the timesteps. For each timestep we extract all the x-positions for all the atoms in a list `xit`. Next, we find a list of all the atoms that are on the left side, that is, all the atoms with an x-position smaller than $L_x/2$. This is done by the `find` command. Finally, we count how many elements are in this list. This is the number of atoms that are on the left side of the box. We store this number in the array `nleft` for this timestep, and reloop to address the next timestep:

```
#Get information about simulation box
tmp_time,box,atoms,bonds,tris,lines = data.viz(0)
halfsize = 0.5*box[3] # Box size in x-dir
for it in range(nt):
   xit = array(data.vecs(it,"x"))
   jj = find(xit<halfsize)
   nleft[it] = size(jj)
```

The complete program reads:

```
#start1
from pylab import *
data = dump("dump.lammpstrj") # Read all timesteps
t = data.time()
nt = size(t)
nleft = zeros(n,float); # Store number of particles
#Get information about simulation box
tmp_time,box,atoms,bonds,tris,lines = data.viz(0)
halfsize = 0.5*box[3] # Box size in x-dir
for it in range(nt):
   xit = array(data.vecs(it,"x"))
   jj = find(xit<halfsize)
   nleft[it] = size(jj)
plot(t,nleft), xlabel('t'), ylabel('N_p'), show()
```

A short and simple program to handle a complicated set of data. This should provide you with a good basis for measuring various properties of the simulated system using Python.

3.4.3 Results

The resulting plot is shown in fig. **??**. If you wonder why the number of atoms on the left is varying, how it is varying, and how to describe how it is varying — this indeed in the topic of this book so just read on!

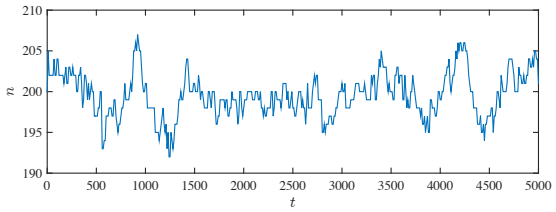

Fig. 3.5 Plot of $n(t)$, the number of atoms on the left side of the simulation box, as a function of time, where the time is measured in the number of timesteps in the molecular dynamics simulation.

3.5 Advanced models

Molecular dynamics simulation packages such as LAMMPS are professional tools that are used for research and development. Here, I provide a few examples of more advanced use of LAMMPS that may inspire you to try the tool also on your own.

3.5.1 Coarsening

Out first example is an extension of the first two-dimensional simulation you performed above. What happens if we start the system with a homogeneous distribution of atoms, but with a low initial energy? In addition, we keep the average kinetic energy in the system approximately constant. (This corresponds, as you will learn later, approximately to keeping the temperature in the system constant):

```
units lj
dimension 2
boundary p p p
atom_style atomic

lattice hex 0.5
region simbox block 0 80 0 40 -0.1 0.1
create_box 1 simbox
create_atoms 1 box

mass 1 1.0
velocity all create 0.05 87287

pair_style lj/cut 2.5
pair_coeff 1 1 1.0 1.0 2.5

neighbor 0.3 bin
neigh_modify every 20 delay 0 check no

fix 1 all nvt temp 0.25 0.25 1.0
dump 1 all atom 1000 dump.lammpstrj
thermo 100
run 50000
```

The resulting behavior shown in Fig. **??** is a phase separation: The system separates into a liquid and a gas phase, demonstrating a phenomenon called spinoidal decomposition. We will discuss the behavior of such systems using both molecular dynamics models and algorithmic models later.

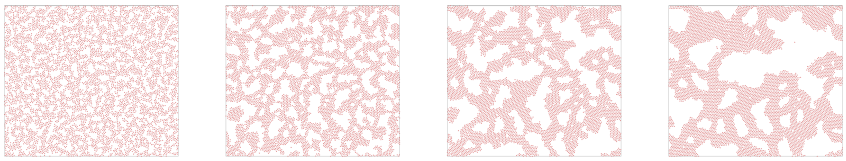

Fig. 3.6 Snapshots from a simulation of a Lennard-Jones liquid, with a low initial density. A slowening coarsening is observed.

3.5.2 Hemoglobin in water

3.5.3 Lipid bilayers

3.5.4 Fracture of Silicon crystal

3.5.5 Water permeation in a carbon nanotube

Summary

- Molecular dynamics simulations model how atomic systems propagate forward in time by integrating the equations of motion based on a given interaction potential between the atoms.
- We have introduced a commonly used interaction potential, the Lennard-Jones potential: $V(r) = 4\varepsilon\left((\sigma/r)^{-12} - (\sigma/r)^{-6}\right)$, where the behavior is determined by the energy scale ε_0 and the length scale σ.
- We run molecular dynamics simulations using custom code, public or commercial codes, which include efficient integrators.
- A typical simulation models a system with constant energy, volume, and number of particles.
- Molecular dynamics simulations are routinely used as part of research and development tasks across disciplines, and are important tools for applications such as the development of new materials, chemical catalysis optimization and drug development.

Exercises

Problems

Exercise 3.1. Evolution of density

We will here study a $10 \times 10 \times 10$ system of Argon atoms, as done above. Initiate the system with all particles in the left 1/3 of the system. We call the number of particles in the left, center and middle thirds of the system $n_1(t)$, $n_2(t)$, and $n_3(t)$ respectively.

a) How do you expect n_1, n_2, and n_3 to develop in time?

b) Measure $n_1(t)$, $n_2(t)$, and $n_3(t)$ and compare with your predictions.

Exercise 3.2. Temperature variation

a) Measure the maximum and minimum temperature, $T_{\max}$ and $T_{\min}$ respectively for a simulation of $L = 10 \times 10 \times 10$ atoms, and calculate $\Delta T = T_{\max} - T_{\min}$.

b) Measure ΔT for three different system size, $L = 4$, $L = 8$, and $L = 16$. Comment on the results.

Exercise 3.3. Coarsening

Start a LAMMPS simulation with the following init file:

```
# 3d Lennard-Jones melt
units       lj
atom_style  atomic
lattice     fcc 0.2
region      box block 0 10 0 10 0 10
create_box  1 box
create_atoms    1 box
mass        1 1.0
velocity    all create 0.1 87287
pair_style  lj/cut 2.5
pair_coeff  1 1 1.0 1.0 2.5
neighbor    0.3 bin
neigh_modify    every 20 delay 0 check no
fix 1 all nvt temp 0.05 0.05 1.0
dump        id all atom 50 dump.lammpstrj
thermo      100
run     10000
```

(This simulation is performed at constant temperature and not at constant energy). Visualize the time development of the system and comment on what you see.