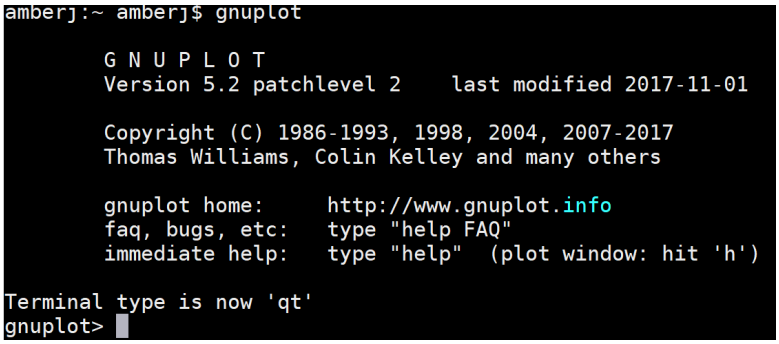
Thursday, 22 August 2019 4:00 PM

Molecular Dynamics Simulations

Resources:
"Understanding molecular simulations: From algorithms to applications" by Frenkel and Smit
"Computer simulation of liquids" by Allen and Tildesley

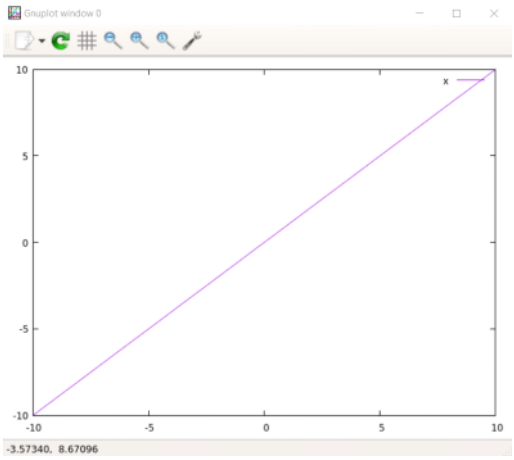
Pre-requisites:
You will need to install gnuplot. For ubuntu, use:
sudo apt-get install gnuplot

To check if installed correctly, type gnuplot on the terminal, and something like this will show up:



Screen clipping taken: 3/23/2020 10:18 AM

Now enter:
gnuplot> plot x
A new screen should show up with a line like the following:

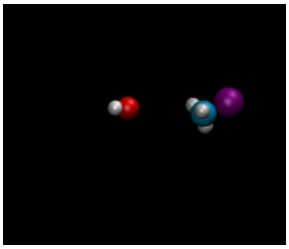


Screen clipping taken: 3/23/2020 10:19 AM

If you get this, success! You are all set with gnuplot. If not, try troubleshooting with the help of internet and email me with the error messages.

What is molecular dynamics (MD) simulations?

Check out http://hase-group.ttu.edu/animations.html for a number of videos and applications of MD



Snapshot of a SN2 reaction from http://hase-group.ttu.edu/group/animations/jingr1.html. For actual animation, go to the website!

Molecular dynamics (MD) is a computer simulation of particles governed by Newton's laws of motion. Consider a box with N particles in it. These particles usually are atoms that constitute the process/reaction that we are interested in. This may include the reactive molecules' atoms as well as any solvent such as water surrounding it. In MD, we assume that the dynamics of these N particles is governed by Newton's laws of motion:

$$\vec{F} = m\vec{a} = m\frac{d^2\vec{x}}{dt^2} \qquad Eq.1$$

This is a differential equation, and MD is a numerical solution to this equation.

To solve the above equation, we need the force $\vec{F}$. We will not worry about to get force for now. Lets assume there is a function provided to you that takes position $\vec{x}$ as input and provides $\vec{F}$ as output. This might come from Hartree-Fock, empirical guess or any other method.

We focus on a different question instead: if such a function $\vec{F}(\vec{x})$ is provided, how does one find a solution to the Newton's laws of motion? First note that Eq. 1 involves a double differential in time. This is usually hard to solve computationally. We instead cast this equation as two single differential equations:

$$\frac{d\vec{x}}{dt} = \vec{v} \qquad Eq.2$$

$$\frac{d\vec{v}}{dt} = \vec{a} = \frac{\vec{F}(\vec{x})}{m} \qquad Eq.3$$

Here $\vec{v}$ is the velocity. In general $(\vec{x}, \vec{v})$ is called the **state of the system**, and contains the entire information about the system. Any property that needs to be calculated, can be derived from this state of the system. Our task hence is to find $(\vec{x}, \vec{v})$ as a function of time. Now, remember that on a computer, we do not have continuous variables. We discretize time in small units. Let the **time step = dt**

Notation: $(\vec{x}, \vec{v})$ - state of the system that contains maximum information about the system

Our task is hence as follows: If I know the state $\left(\vec{x}(t_0), \vec{v}(t_0)\right)$ at some time t_0, how to obtain the state $\left(\vec{x}(t_0 + dt), \vec{v}(t_0 + dt)\right)$ at time $t_0 + dt$? Believe it or not, this is a non-trivial question and was researched heavily upon for almost a decade. Let us start with a simple guess - Fourier expansion:

$$\vec{x}(t_0 + dt) \approx x(t_0) + \vec{v}(t_0)dt + \frac{1}{2}\vec{a}(t_0)dt^2 \qquad Eq.4$$

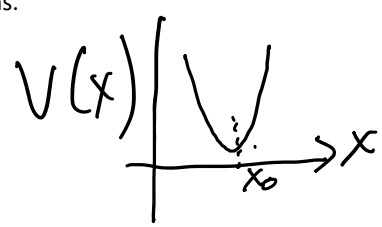
$$\vec{v}(t_0 + dt) \approx v(t_0) + \vec{a}(t_0)dt \qquad Eq.5$$

Here we have used Eqs. 2 and 3. (Side note: we have truncated Eq. 5 to first order in dt since velocity is itself a first order term in dt. In any case, finding the second order term in dt in Eq. 5 requires finding derivative of acceleration with time, which is not computationally feasible.)

Equations 4 and 5 provide us with a way to evolve our state of the system now from $\left(\vec{x}(t_0), \vec{v}(t_0)\right)$ to $\left(\vec{x}(t_0 + dt), \vec{v}(t_0 + dt)\right)$! We will test these equations in the next section. However, be warned, this is not the best method, but a trial method to get us started.

Let us start with our first MD code!

To see how Eqs. 4 and 5 work in practise, we will write a basic MD code in Python for a simple system. Consider a diatomic molecule: H_2. For starting, we will consider only its vibration, and no rotation or translations.



Let 'x' denote the distance between the two H-atoms, and let 'x_0' be the equilibrium bond length. Now bonds can often be thought of as springs, or a simple harmonic oscillator. We approximate the potential that this bond feels as

$$V(x) = \frac{1}{2} m\omega^2 (x - x_0)^2, \qquad Eq.6$$

where m is the mass, and ω is the frequency of the bond. This gives force F as:

$$F = -\frac{\partial V}{\partial x} = -m\omega^2 (x - x_0) \qquad Eq.7$$

Now, as it so happens, the evolution of the state of the system $(\vec{x}, \vec{v})$ for a simple harmonic oscillator can be solved analytically. You can look up the solution, or solve it yourselves. However, for a more realistic potential, such as Lenard-Jones or Morse potential, one cannot solve it analytically. Our purpose here is to solve these equations numerically for a generic potential - SHO is our play tool.

Let's think about the inputs and outputs of our first MD code.

Output of the code:
The output is (x, v) as a function of time in steps of dt.

Input to the code:
Now, what will constitute our inputs? Well, the time step dt and total simualtion time T are 2 inputs. Can you guess what other inputs we need to give? We also need to provide the parameters for our potential: mass m, frequency ω and equilibrium bond length x_0. Finally, we also need to give the initial condition x(t=0).

Required parameters
Let's estimate m, ω and x_0. For H_2, what are some reasonable values?

➢ Do not read any further - first make estimates for m, ω and x_0 by yourself before proceeding.

Now that you have made estimates, let's compare how you did. Remember, here we are making ballpark estimates - exact number is not important for now. Let's take mass to be mass of proton: m = 1837 a.u. (atomic units), and frequency = 4000 cm^{-1}. The bond length of H_2 is about 0.7 angstroms or 1.4 a.u.

➢ Convert 4000 cm^{-1} to atomic units (in s-1). Hint: you will need values of pi and speed of light to convert.

Next, let us look at dt and T. Since we already know that the position will be oscillating with a time period of $2\pi/\omega$, our time period should be much smaller than this time period: dt $\ll \frac{2\pi}{\omega}$. We will later look at how to get a converged estimate for dt. For now, we will take dt $= \frac{1}{50}\frac{2\pi}{\omega}$. The total simulation time T depends on the problem of interest and we will discuss that in more detail later. For now, to be ale to see some oscillations, we will choose T $= 20\frac{2\pi}{\omega}$.

To summarize our parameters are:

Mass m	1837
Frequency ω	4000 * wavenumbers_to_au
Equilibrium bond length x_0	2
Time step dt	$\frac{1}{50}\frac{2\pi}{\omega}$
Simulation time T	$20\frac{2\pi}{\omega}$

Initial Conditions
Finally, we need to provide initial conditions. In general, initial conditions depend on the problem we are studying. For example, if we were simulating a system at thermal equilibrium, initial conditions will be given by Boltzmann distribution. More on this later. For starters, we will choose:

$$x(t = 0) = x_0 + 1(a.u.)$$
v(t=0)=0.

Algorithm:
There are 3 steps to the MD simulation:

1. Setting up parameters as provided above.
2. Setting initial conditions: x(t=0)=$x_0 + 1$; v(t=0)=0
3. Evolution:
   a. Find number of time steps required: nsteps = int(T/dt)
   b. Create a for loop for the number of time steps: for i in range(nsteps):
   c. Update (x, v) using Eqs. 4 and 5 to evolve one time step at a time.

d. Calculate force (acceleration) for the next time step.
e. Remember to print at each time step

- One nuance to be careful about - at the very first time, one needs to calculate acceleration before the loop begins.
- In a general MD simulation, there are 2 additional steps to the algorithm: (a) equilibrating before evolution, and (b) calcul ating the observable of interest during the evolution or post-evolution depending on the problem.

To action!

- [ ] Now, move onto the provided partial code md.py. This code has a few blanks you need to fill in. Take your time, and fill in all the blanks.

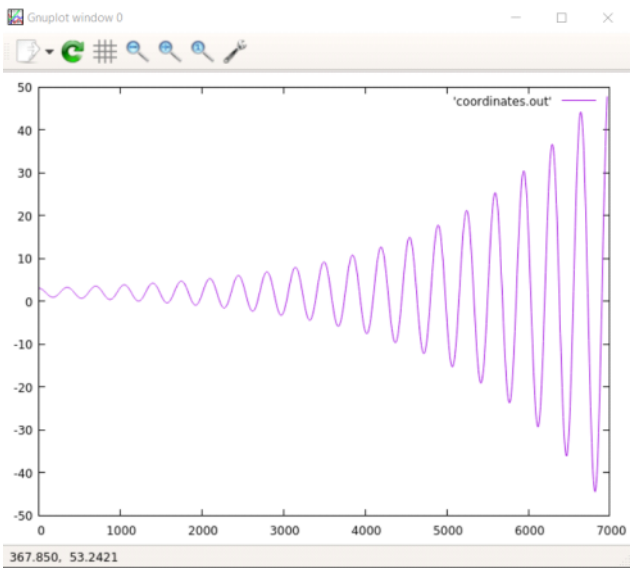
On successful run, you should get a file coordinates.out, that contains the output. Compare with the provided coordinates_benchmark.out to make sure you get the same output.

Let us plot position as a function of time using gnuplot.

- [ ] On terminal, type gnuplot and press enter. Now type:
- [ ] plot "coordinates.out" with lines

```
amberj:molecular_dynamics amberj$ gnuplot

        G N U P L O T
        Version 5.2 patchlevel 2    last modified 2017-11-01

        Copyright (C) 1986-1993, 1998, 2004, 2007-2017
        Thomas Williams, Colin Kelley and many others

        gnuplot home:     http://www.gnuplot.info
        faq, bugs, etc:   type "help FAQ"
        immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> plot 'coordinates.out' with lines
```

This will open a new window with the plot.



Screen clipping taken: 3/23/2020 9:52 AM

You can plot yours as well as the provided coordinates_benchmark.out by entering the following on a new line on gnuplot:
plot "coordinates.out" with lines, "coordinates_benchmark.out" with lines

- [ ] What are your observations? Does this plot look correct?

This plot cannot be correct; the intensity of the position should not be growing as a function of time. We should get a cosine like function. What went wrong?

Debugging MD:
One of the most important debugging tools in MD is to look for energy conservation. Energy is given by

$$E = \frac{1}{2}mv^2 + V(x)$$

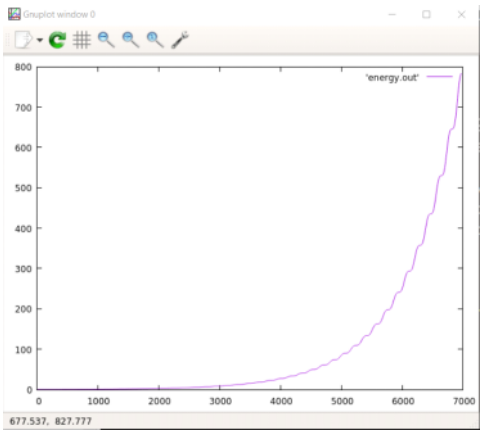
Within the for loop in the evolution step right after the statement

```
pot,acc=compute_pot(pos)
```

calculate energy as

```
energy = pot + 0.5*mass*v*v
```

Now write this energy also in a separate file called energy.out. (You need to open another file with command file_name = open('energy.out', 'w'), and then print in this new file.) Plot energy vs. time using gnuplot. The output is provided in file energy_benchmark.out for you to compare. Compare and make sure you get the same result.



Screen clipping taken: 3/23/2020 10:00 AM

Clearly energy is not conserved. The problem here is the evolution scheme we have used in Eqs. 4 and 5. There is a slightly modified version of this scheme that is much more stable, called the velocity-Verlet:

$$\vec{x}(t_0 + dt) \approx x(t_0) + \vec{v}(t_0)dt + \frac{1}{2}\vec{a}(t_0)dt^2 \qquad Eq.8$$

$$\vec{v}(t_0 + dt) \approx v(t_0) + \frac{1}{2}\big(\vec{a}(t_0) + \vec{a}(t_0 + dt)\big)dt \qquad Eq.9$$

Note that the position evolution remains the same as before. For velocity evolution however, we are instead now using an average of two accelerations at times t_0 and $t_0 + dt$. Almost by magic, this little change improves our evolution scheme dramatically.

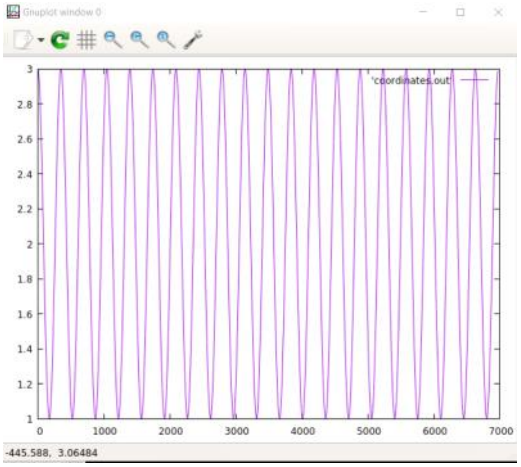
Equations 8 and 9 are called the velocity-Verlet scheme. It is a commonly used schemes in MD simulations. In this course, we will be consistently using this scheme.

Implement Eqs. 8 and 9 in the code. And replot energy and position as a function of time.

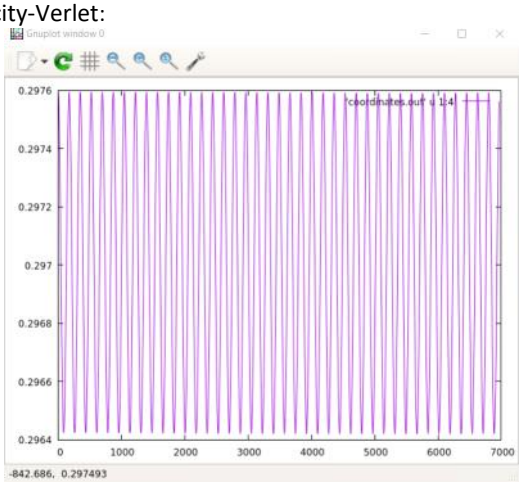
HINT: You should calculate potential or acceleration only once per time step. Calculating acceleration is the most computationally expensive part of the code.

The data for position and energy is provided in the file coordinates_vel_ver.out. The columns are: time, position, velocity, energy. Make sure you can match this file before moving forward.

Position as a function of time for SHO using velocity-Verlet:

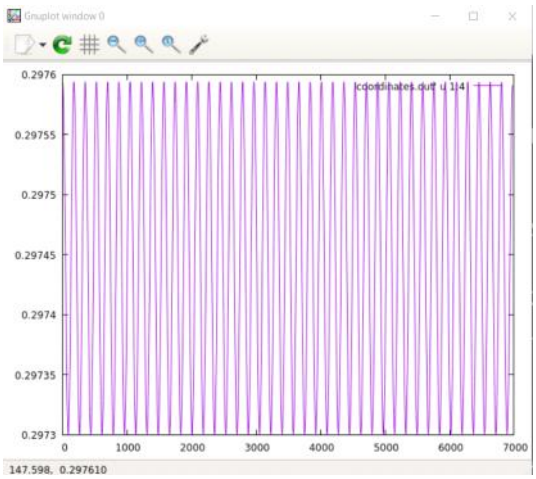


Energy as a function of time for SHO using velocity-Verlet:



Note the immense improvement velocity-Verlet method provides! The energy is far better converged. A few things to note:
1. Energy can never be perfectly constant in a computer simulation. This is an artefact of a finite dt. To test this, re-run the simulation with a smaller dt and plot energy vs. time again. Now the deviation of the energy will decrease.

Energy vs time for dt=$\frac{1}{100}\frac{2\pi}{\omega}$. Note the y-range of this plot and the plot above with dt=$\frac{1}{50}\frac{2\pi}{\omega}$

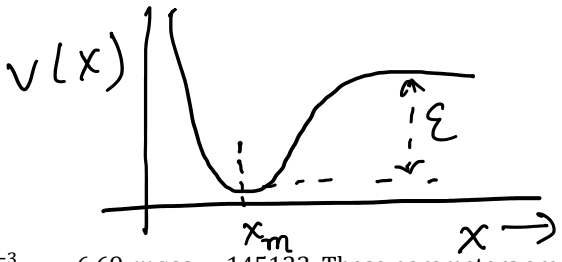


2. Velocity Verlet falls into a category of methods called symplectic integrators. What this means is that in this method, energy will always fluctuate about a value and does not have a shift over time as long as dt is small enough (more on converged dt later).

FINAL TASK: Run the calculations for a Lennard-Jones potential instead of SHO

$$V(x) = 4\epsilon\left(\left(\frac{\sigma}{x}\right)^{12} - \left(\frac{\sigma}{x}\right)^{6}\right)$$

Lennard-Jones is a more realistic potential than harmonic oscillator



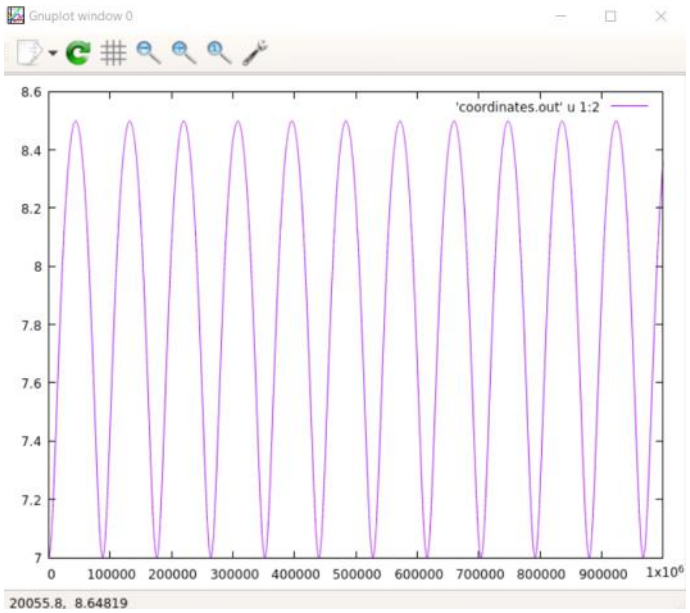
$x_m = (2^{1/6})\sigma$

Use the parameters (in atomic units): $\epsilon = 8.71\text{X}10^{-3}, \sigma = 6.69, mass = 145123$. These parameters are taken for Br2 potential. Use initial conditions (in atomic units):

$$x = 7.0$$
$$v = 0.0$$

Hints:

- The parameters ω, and x_0, are no longer relevant. Replace lines where these two parameters are defined by defining ϵ and σ instead. You will need to provide values of dt and T too. Think what might be reasonable values.
- Calculate the derivate of potential with respect to x on pen and paper. Change the function compute_pot(x) appropriately. Re -run the calculation.
- Plot of position vs. time with dt=100 (a.u.) and T=1.e6 (a.u)



Screen clipping taken: 3/23/2020 11:21 AM

The file coordinates_LJ has the raw data for comparison (columns: time, position, velocity, energy).