Homework 3

Due 2019 October 11

Report instructions

Please upload *one* PDF file to Gradescope (Entry code: ME62YG).

For this homework you will implement code (on PrairieLearn) to perform molecular dynamics simulations using periodic boundary conditions. Specifically, you will reuse (and extend) code from the previous homework and then perform molecular dynamics simulations, as explained in detail below; please see PrairieLearn for details and instructions.

Measuring Physical Properties

Now that you have a functional molecular dynamics code, you want to add the capability to measure important physical and statistical properties of your system. In the previous homework, you already added code to measure the system energy. Now we will implement other observables, including several commonly studied correlation functions. (**Refer to Section 4.4 of Frenkel & Smit, p. 84.**)

Use the same parameters for your Lennard-Jones molecular dynamics simulations:

```
T0 = 0.728      # temperature
L = 4.2323167 # box length
N = 64         # number of particles
M = 48.0       # mass of each particle
h = 0.032       # time step size
steps = 1000   # number of time steps
```

Total Momentum. Write a function for your code (not on `prairielearn`) that measures and outputs the total momentum of your system as it evolves in time.

Temperature. You already have a function for the kinetic energy, which is directly proportional to the instantaneous temperature T, defined by the relationship

$$\sum_i \frac{1}{2} M v_i^2 = \frac{3}{2} N k T$$

where the quantity on the left side is just the kinetic energy. N is the total number of particles. In the reduced units we've been working with, the Boltzmann constant $k = 1$. Add code to output the instantaneous temperature and output it along with the kinetic energy (This is a simple modification to the kinetic energy observable, but when we work with thermostats, it will be useful to be able to study how the energy of your system evolves.).

Pair Correlation Function. The pair correlation function, also known as $g(r)$ or radial distribution function (RDF), is a staple of condensed matter physics. It characterizes the spatial distribution of particles in your system and thus provides a signature of the spatial structure. A typical way to

define the pair correlation function is

$$g(r) = \left(\int dr' \, \Omega(r') \right) \left\langle \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \delta(r - r_{ij}) \right\rangle$$

where r_{ij} denotes the distance between the particles i and j, and we sum over all $N(N-1)/2$ pairs in the system. $\Omega(r)$ denotes the normalization, the exact form of which you will have to determine. Conventionally, $g(r)$ is normalized such that $g(r) = 1$ in a system with randomly distributed particles (i.e. there is no spatial correlation among particles).

This means the normalization is a function of radial distance r. Think about computing unnormalized $g(r)$ for an infinite system of randomly distributed particles, and convince yourself that the number of particles a distance r away will grow in proportion to the volume of a shell (dr) of radius r. Thus, to impose $g(r) = 1$ for a randomly distributed system, we must include in the normalization a term that compensates for this, i.e. that is inversely proportional to the volume as a function of r.

The crux of this task is understanding how to implement $g(r)$ in a discrete, numerical setting. That is, you will not literally perform the integral and you will not end up with $g(r)$ as a continuous function of r. Instead, you want to generate a histogram of $g(r)$. The idea is to look at the distance between each pair of particles, r_{ij}, at a given MD time step. You should declare an array with each element initialized to 0, and you need to define a distance dr which is your grid resolution. Each element i of the array represents the range of distances from idr to $(i+1)dr$. Then for each pair distance r_{ij}, you should "bin" that distance by identifying which array element i contains this value of r_{ij} and then increasing the value of that element by one. **See p. 86 of Frenkel & Smit for an implementation of this function.**

You may find it easier to store all the position coordinates from your simulation and compute the pair correlation after the simulation is complete.

Structure Factor. Next, implement a calculation of the structure factor:

$$S(\mathbf{k}) = \frac{1}{N} \langle \rho_{\mathbf{k}} \rho_{-\mathbf{k}} \rangle \,, \quad \text{where} \quad \rho_{\mathbf{k}} = \sum_{j=1}^{N} e^{-i\mathbf{k}\mathbf{r_j}}$$

and the summation is over all particle position vectors. $S(\mathbf{k})$ can be cast as a Fourier transform of the pair correlation function $g(r)$, and ρ_k is the spatial Fourier transform of the number density. In a periodic box, we may only use wave vectors $\mathbf{k}$ that are commensurate with the periodicity of the box, i.e.

$$\mathbf{k} = \frac{2\pi}{L}(n_x, n_y, n_z), \quad \text{where} \quad n_x, n_y, n_z = 0, 1, 2 \ldots$$

Write a function `LegalKVecs(maxK)` that computes a list of the legal k-vectors. `maxK` is an integer specifying the largest value of (n_x, n_y, n_z) that is allowed. The larger k-vectors you use, the smaller length scales you can probe. However, the number of k-vectors you sample scales as `maxK`^{3}, so you should start by choosing `maxK=5`.

Then write two functions `rhoK(k)` and `Sk(kList)`: The first one computes the Fourier transform of the density for a given wave vector, ρ_k from the formula above. The second one computes the structure factor for all k vectors in `kList` and returns a list of them.

You should then be able to plot your results in the following way:

```
kmags  = [np.linalg.norm(kvec) for kvec in kvecs]
sk_arr = np.array(sk_list) # convert to numpy array if not already so

# average S(k) if multiple k-vectors have the same magnitude
unique_kmags = np.unique(kmags)
unique_sk    = np.zeros(len(unique_kmags))
for iukmag in range(len(unique_kmags)):
    kmag    = unique_kmags[iukmag]
    idx2avg = np.where(kmags==kmag)
    unique_sk[iukmag] = np.mean(sk_arr[idx2avg])
# end for iukmag

# visualize
plt.plot(unique_kmags,unique_sk)
```

Velocity-velocity correlation. Add an observable to your code that measures the velocity-velocity correlation function:

$$\langle \mathbf{V}(0) \cdot \mathbf{V}(t) \rangle$$

You should compute this expectation value by averaging over each particle. You will have to store

$$N_{\text{stored}} = \frac{t_{\text{max}}}{dt}$$

previous velocities, where t_{max} is the largest value of t for which you want to compute this correlation.

Diffusion constant. Add an observable to your code that calculates the diffusion constant, which is simply an integral over the velocity-velocity auto-correlation function,

$$D = \int d\tau \ \langle \mathbf{V}(0) \cdot \mathbf{V}(\tau) \rangle$$

(You may alternatively store all the velocities from your simulation and compute the velocity-velocity correlation after the simulation is complete.)

Canonical Ensemble and Thermostatting. So far, we have performed molecular dynamics in the micro-canonical ensemble (constant energy). We are now going to modify our molecular dynamics code to work in the canonical ensemble (constant temperature). There are a number of ways to modify the code to simulate a canonical ensemble; generally this involves coupling the system to a heat bath which allows it to sample different energy states. Some methods are simpler than others and some methods are more accurate than others. We will start by implementing the Andersen thermostat, which is the most straightforward algorithm to implement. This method ensures that you will get the correct properties for static equilibrium observables (e.g. pair correlation functions, energy, etc.) but the dynamics are artificial and you should not expect your simulation to provide reliable information about dynamical properties of the system (e.g. diffusion constant).

Andersen Thermostat. Algorithm:

- Start with an initial set of configurations and momenta and integrate for Δt (your code already does this).
- Loop over all particles and for each one, select it to undergo a collision with probability $\eta \Delta t$ where Δt is the time step and η is a parameter defining the coupling strength to the bath (you should choose the value of η so you get a collision probability around 1%).
- For each particle that is selected to undergo a collision, its new velocity is drawn from the Maxwell-Boltzmann distribution (i.e., select a new velocity for each component from a Gaussian distribution with $\sigma = \sqrt{T/M}$ and mean 0.

You may want to refer to the documentation on Python's `random` module to see how to generate random numbers drawn from a Gaussian distribution. (You can implement this algorithm by adding about 5 lines of code to your existing MD code). You can also refer to Frenkel & Smit, p. 140 for a discussion and implementation.

Your report

1. Is momentum conserved? Do you expect it to be conserved?
2. Produce a plot of $g(r)$ for the Lennard-Jones simulation conditions described above! If your system takes time to equilibrate from its initial conditions (i.e., if you see a transient in your energy trace), you should not collect $g(r)$ data until your system is well equilibrated.
3. Produce a plot of $S(k)$ from a run using the Lennard-Jones system parameters described above. Be sure to normalize your function correctly! Describe qualitatively how $S(k)$ and $g(r)$ should look for a liquid and a solid.
4. Produce a plot of the velocity-velocity correlation as a function of time. Also report your value for the diffusion constant.
5. Produce a plot of the temperature with and without using the Andersen thermostat. Then increase the temperature to `T=1.0`, but still pass the value `0.728` as an argument to the `InitVelocity` function. Re-run and produce a plot of temperature vs. time for this simulation.
6. A limitation of the Andersen algorithm is that it uses randomized velocities, which corrupt the dynamics of the system. Static equilibrium properties are still accurate, but you will no longer be simulating the physical dynamics of your system. To see this limitation of the Andersen thermostat quantitatively, adjust the value of η so that your collision probability is around 50%. Produce a plot of the velocity-velocity correlation function with thermostatting on and report your calculation of the diffusion constant. Compare these results with your data from the micro-canonical simulation (above). Specifically, you should discuss what differences you expect to see in these observables and whether or not you do, in fact, see them.
7. In this problem, you will be using your molecular dynamics code to estimate the transition temperature at which argon turns from a solid to a liquid. To begin, let's specify the relevant parameters for our argon simulation, which again uses the Lennard-Jones potential in reduced units (i.e., $\varepsilon = \sigma = 1$):
    - mass: 48.0
    - density: 1.0
    - timestep: 0.032
    - particle number: 64

- using Andersen thermostat for canonical ensemble
- myTemp = 2.0

Choose 5 temperatures in the range between 0 and 4 and run MD simulations at those temperatures. From this, you should be able to roughly identify the location of the melting/freezing transition. Now, try to refine your estimate by choosing three more temperature points in this vicinity. You should study all of the observables you have implemented to determine the phase of your system at a given temperature. Examining the structure factor is likely the most useful way to determine the phase of your system. Submit your estimate of the transition temperature along with data that support this claim. Minimally, you should include graphs of the pair correlation function and structure factor for temperatures above and below your estimated transition temperature.