Exercise 4

Due: Thursday, 11/7/19

Objective: To perform a basic Monte Carlo simulation, and to compute thermodynamic property averages from it.

Consider the Lennard-Jones / harmonic spring model of a linear polymer that you examined in exercise 3. In this assignment, you will write a Monte Carlo simulation based on that model. You can use your old codes for exercise 3 as templates for writing new ones in this exercise.

Part a

Write a Python code and an accompanying Fortran library to perform a Monte Carlo simulation of the polymeric system. Your Monte Carlo moves should consist of single-atom displacements by random amounts in each of the x, y, z directions. You may want to define a separate function that performs such a move (picking a random particle and making random displacements) and accepts or rejects it based on the Metropolis criterion.

In Python, you will need to keep in mind that explicit copies of arrays are required to save positions before moving a particle, due to name binding. The approach might look something like:

```
MoveAtom = np.random.randint(N)
OldPos = Pos[MoveAtom,:].copy()
#compute old energy here
#modify Pos[MoveAtom,:]
#compute new energy
#decide to accept/reject
if Reject:
    Pos[MoveAtom,:] = OldPos
```

The **`copy()`** command is required because, otherwise, **`OldPos`** will point to the same data in memory as the corresponding subsection of **`Pos`**. Therefore, modifications to **`Pos`** would then also affect **`OldPos`** unless a copy is made.

As with the last project, your simulation will take place under periodic boundary conditions in a cubic box of side length L. For the nonbonded interactions, you should cut and shift the Lennard-Jones potential at a distance of $r_c = 2.5$. Do not apply a tail correction.

Keep in mind that you do not need to compute the entire potential energy every time a single atom is moved. Rather, you can compute only the energy of that atom with all other atoms. You will have to do this twice in order to get the change in potential energy ΔU: once before the atom is moved and once after. Therefore, add an additional Fortran function to your Fortran file that computes the total interaction energy *due to a specified atom*.

You can keep a running total of the energy by doing updates of the form $U \leftarrow U + \Delta U$. However, due to precision issues, it is always a good idea to do a full update of the energy (the entire pairwise loop) once in a while, say every 10-100 or so moves per particle (e.g., every 1000-10000 moves for a 100 particle system). Moreover, it is important not to do the update $U \leftarrow U + \Delta U$ until *after* a move has been accepted. Otherwise, a high-energy ΔU from a rejected move could overwhelm U and cause precision errors. More is on this issue in the handout "Simulation best practices".

During the MC steps, you should also compute the average acceptance ratio by keeping track of the numbers of proposed and accepted moves.

Part b

Enable your code to compute the virial and, hence, the pressure. This will require the addition to the Fortran pairwise interaction loop a summation over virial terms. For single-particle displacements, one only needs to compute the change in the virial due to the displaced particle, similar to the potential energy. For pair interactions u_{ij}, the relevant equations are:

$$P = \frac{Nk_BT}{V} - \frac{\langle W \rangle}{3V} \qquad\qquad W = \sum_{i<j} \frac{du_{ij}(r_{ij})}{dr} r_{ij}$$

Part c

Consider the cases $M = 1,2,4$ at the following conditions:

- $N = 240 \quad (N \text{ is the number of monomers})$
- $\rho = N/V = 0.9$ so that $L = (N/\rho)^{1/3}$
- $T = 2.0$

Develop a run protocol in your code:

- Initially place atoms on a cubic lattice.
- Energy-minimize the initial configuration using the conjugate-gradient method.

- Perform Monte Carlo steps.

First, perform initial test runs for the cases $M = 1,2,4$. Find a value of the maximum displacement in each case that gives ~50% acceptance. How does the optimum maximum displacement vary with chain length?

Next, perform a longer run using the maximum displacements determined above. On a single graph, plot for the three cases their potential energy U as a function of the number of "sweeps" — the number of MC attempts or steps divided by the number of particles. Identify a time, in sweeps, at which the system appears to have equilibrated. Indicate this on your graph. How does the equilibration time vary with chain length?

Part d

Using the equilibration times identified above, perform a new set of simulations and compute $\langle P \rangle$ at four different densities, $\rho = 0.75, 0.80, 0.85, 0.90$, for each of $M = 1,2,4$. You will have to choose a suitable production run time in order to gain sufficient statistics. You may even want to run multiple trials for averaging. You should start collecting data for the pressure only after equilibration. Prepare a plot of the equation of state for each case $M = 1,2,4$ that gives the pressure as a function of density. Include error bars on your pressure calculations. Does polymerization increase or decrease the pressure? (Does this make sense physically?)

Part e – ADVANCED TRACK

Generally, MC simulations with single particle displacements are less efficient than MD for molecules with many stiff bonds. By efficient, we mean the rate at which the simulation explores configuration space, which we might measure by a mean squared displacement. To make a fair comparison, we need to examine how far the system moves as a function of equivalent computational effort: either one MC *sweep* or one MD step. From the production periods for $\rho = 0.90$ in part d and for $M = 1,2,4$, compute the mean-squared displacement as a function of MC sweeps. Using your code from exercise 3, also compute the mean squared displacement as a function of MD steps for the same conditions. Make a graph with these six curves and compare the rate at which the two methods explore configuration space.

What to turn in

For the assignment, submit a short, typed summary of your results, clearly indicating the problem components. In addition, print a copy of your Python code for reference and attach it to the back of these.