Task

The aim of the task is to program a basic molecular dynamics (MD) program within the classical picture (no quantum mechanics, no relativity theory). The Newton equations of motions should be integrated numerically for a certain number of finite time steps.

Consider a set of *N* particles with mass *m* at finite temperature *T*. Assume the simulation box is two-dimensional, meaning that the particles have coordinates (*x*,*y*). The particles have pairwise interactions. Each pairwise interaction term is modeled by the van der Waals interaction (Lennard-Jones),

$$E_{ij} = 4\epsilon \left(\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right)$$

where r_{ij} is the distance between particle *i* and *j*, σ is a measure for the size of each particle, and ε is a measure for the interaction strength between the particles. The sum of the pairwise interactions is the potential energy of the system. Moreover, each particle has kinetic energy, and the sum of the kinetic energies of each of the particles is the kinetic energy of the system.

The simulation box has a square shape (side length 20 Angstrom). The boundaries of the box are located at the x_i=0, x_f=20 Å, y_i=0, y_f=20 Å and are assumed to be reflective*. In practice, this means that, when an integration step would put the particle outside the box, then the particle is reflected back into the box as if it bounced on the wall.

The parameters below are chosen to simulate 16 argon atoms (*N*=16) inside the box:

m = 39.948 g/mol
T = 300 K
σ = 3.4 Å (Angstroms)
ε = 0.24 kcal/mol

However, because of the simplifications that are considered in this task such as:
* Two-dimensional system,
* No control over temperature/pressure,
* Non-periodic boundary conditions,
the exact properties of argon gas are not likely to be seen.

The steps for the simulation are as below:

1) Read or copy the initial coordinates and velocities from the file Ar_initial.txt. In each section of the file, the first column shows the atom ID, the second column is the *x* component and the third column is the *y* component of the velocity or position.

2) You need to compute the force (~ derivative of the Lennard-Jones relationship) on each atom caused by all other atoms in the system due to their positions. Positions, velocities, accelerations, and forces are assumed to be all classical quantities. Print out the forces on the atoms for the given initial coordinates.

3) Write an integrator of the Newton equations of motion using an appropriately chosen finite time step Δt. Perform at least 10000 integrations. It is recommended to use the velocity-Verlet algorithm, which is shown below:

$$\mathbf{x}(t + \Delta t) = \mathbf{x}(t) + \mathbf{v}(t)\,\Delta t + \tfrac{1}{2}\,\mathbf{a}(t)\Delta t^2,$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \frac{\mathbf{a}(t) + \mathbf{a}(t + \Delta t)}{2}\Delta t.$$

You have now programmed a very short MD simulation.
Suggestion: write a function that takes current positions and velocities at time step *n* and returns the positions and velocities at time step *n*+1.

4) For every 100 steps, print out the potential energy, the kinetic energy, instantaneous temperature, and the total energy (potential+kinetic) of the system and plot them as a function of time. Energies can be expressed per mol (also in your code).

5) Plot the trajectory in order to observe the motion of the argon atoms in the box. Concretely, plot the position of the particles every 100 steps.

* reflective boundary condition (from google):

“At each time step in your molecular simulation, you're updating the velocities and positions of your particles. If the updated position of one of your particles crosses a boundary, you can implement reflective boundary conditions by flipping the sign on the velocity component orthogonal to the boundary, as if the particle had undergone an elastic collision with a wall. For instance, if one particle is at x=0.1 moving with vx = -0.3, and I have a boundary at x=0, when I step forward in time (we'll say Delta t = 1) my new x-coordinate would be -0.2. This crosses my boundary, so I'll instead update the x-position to 0.2, and then change the x-velocity to +0.3. If you have difficulties with the implementation, just remember it's like an elastic collision and you can work out what the new position and velocity should be using classical mechanics.”