Computer simulation of physical systems I

Task II: NVE molecular dynamics simulations

In this exercise, you are going to perform molecular dynamics (MD) simulations of a system of atoms interacting through a Lennard-Jones pair potential in the NVE statistical ensemble. To complete this exercise you should download the python version of the MD code available on the website and closely follow the instructions provided in this guide.

All the functions necessary for performing MD simulations are implemented in `MD.py`, which has to be imported in your python code, i.e. `from MD import *`.

1. *Startup*: generate the starting atomic positions, the starting velocities, then change the system temperature to the desired value and equilibrate the sample.

- Use the function `crystal()` to generate atomic positions on a FCC lattice; choose the parameters to match the number of atoms ($N = 864$) and density ($\rho = 1.374\,\mathrm{g}\cdot\mathrm{cm}^{-3}$) for liquid Ar as in Rahman's paper[1] (see also additional notes).
- Perform a short run to compute the velocities and then use the constant velocity rescaling to adjust the temperature to the value that you can find in Rahman's paper ($T \simeq 94\,\mathrm{K}$).
- Equilibrate the system with a run in free evolution mode (regular NVE dynamics); monitor the conservation of energy and the fluctuations of other quantities.
- (Optional): check the conservation of energy as a function of the integration step Δt; study the dependency of the fluctuations on the system size (i.e., as a function of N).

2. *Sample static properties*: sample the radial pair correlation function, $g(r)$, and also the structure factor, $S(k)$, computed as the radial Fourier Transform (FT) of $g(r)$. Compare the position of the peaks with those reported in Rahman's paper. Try to explain the behaviour of $g(r)$ and its limits for $r \to 0$ and $r \to \infty$.

3. *Sample dynamical properties*: estimate the diffusion coefficient, D, using two different methods, namely

(a) Einstein's relation, $\mathrm{MSD}(t) = C + 6Dt$, which involves the sampling of the mean square displacement (MSD) and a fitting procedure giving C and D

(b) the time integral of the velocity autocorrelation function

4. (Optional) Repeat the same steps for a system at a different temperature and/or density (remember that the accuracy of the time integration may depend on these parameters) and compare the static and dynamic properties.

[1] A. Rahman, Phys. Rev. A **136**, A405 (1964).

Task II: additional notes on MD units

The MD codes provided for this exercise (and the next one) solve numerically the equations of motion for a Lennard-Jones (LJ) fluid made of N particles contained in a 3D periodically repeated rectangular box and interacting through the pair potential:

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] .$$

The codes use internally the so-called LJ units (i.e. σ and ε are both set to 1), so that distances are expressed in σ units, while energies are given in ε units. Therefore, once the parameters of the the LJ potential have been fixed (for instance, $\sigma = 3.4\,\text{Å}$ and $\varepsilon/k_B = 120\,K$ for Ar, see Rahman's paper) the units of measure for all physical quantities are also fixed, as listed in the following table for those of our interest:

quantity	LJ units
distance	σ
energy	ε
velocity	$(\varepsilon/M)^{1/2}$
temperature	ε/k_B
time	$\sigma(\varepsilon/M)^{-1/2}$

where M is the mass of an atom in the simulation (for instance, $M \simeq 6.69 \cdot 10^{-26}$ kg for Ar).

As an example, we can use the relations above to convert the integration time step used in Rahman's paper ($\Delta t = 10^{-14}$s) from seconds to LJ units. Inverting the relation

$$\Delta t = \sigma(\varepsilon/M)^{-1/2} \cdot \Delta u$$

to obtain Δu (in LJ units), we get the following value (expressing all parameters in SI units):

$$\Delta u = \Delta t \cdot (\varepsilon/M)^{1/2}/\sigma = 10^{-14} \cdot \left(\frac{120 \cdot 1.38 \cdot 10^{-23}}{6.69 \cdot 10^{-26}} \right)^{1/2} \cdot \frac{1}{3.4 \cdot 10^{-10}} \simeq 0.0046 ,$$

which will be employed in our simulations in order to compare with Rahman's results.

Instructions

- Download and unpack the archive `Task2.zip`. The MD code is implemented in `MD.py` and the files `Step*.py` contain minimal scripts required to complete the step.
- if you managed to get the code working on your workstation, then you are ready to start: change directory to `Step1_Startup` and move to Step 1.

MD code

The MD code requires the following libraries:

- `numpy` for linear algebra (https://numpy.org)
- `matplotlib` for plotting (https://matplotlib.org)
- `scipy` for statistics analysis and fitting (https://scipy.org)
- `numba` for acceleration of the most heavy functions (http://numba.pinata.org). The use of this library can be easily avoided if necessary, but be prepared to loose the performance. To eliminate `numba` from the code, remove the line `@jit(parallel=True)` In `MD.py`.

The main functions used for running MD simulation are `run_NVE()` and `run_NVT()`. These functions return all the results in a dictionary, which contains energies, velocities, $g(r)$ etc. You should find all the possible output in the return statement of the corresponding function.

Step 1: Startup

1. Running `crystal()`

First of all, you must generate a configuration to start from. The function `crystal()` is used to arrange atoms in a crystalline fcc structure.

The function `crystral()` takes two arguments: the number of units fcc cells along each direction `Ncells` and the lattice spacing `lat_par`.

The number of unit fcc cells (containing 4 atoms each) to stack along the three directions: choose them in order to get a cubic box with same number of particles ($N = 864$) used in [1], hence select 6 unit cells along each axis so that N will be equal to $4 \times (6 \times 6 \times 6) = 864$ (in general you should not put less unit cells than what suggested to satisfy the minimum image criterion, but 6 cells is more than enough in this example). This number of cells, combined with the lattice parameter chosen above, gives a box size approximately equal to that in [1] ($L = 10.229$ in L.J. units, please see the notes), so that the densities will be the same too.

The lattice spacing of the fcc crystal is the equilibrium lattice spacing of the LJ potential is 1.5496, but here we choose a value, $a = 1.7048$, that corresponds to the density studied by Rahman [1], i.e. $1.374\ \mathrm{g} \cdot \mathrm{cm}^{-3}$ for Ar (with atomic mass approx. $M = 6.69 \cdot 10^{-23}\mathrm{g}$).

The function `crystal()` returns two arrays: coordinates and velocities, the latter assigned randomly according to Gaussian distribution.

The simplest example of using `crystal()` is provided in `Step1.py`:

```
Ncells = 6          # Number of unit cells along each axis
lat_par = 1.7048    # Lattice parameter
L = lat_par*Ncells  # Size of the simulation box
N = 4*Ncells**3     # Number of atoms in the simulation box

# Generate fcc structure
pos, vel = crystal(Ncells, lat_par)
```

2. Running the MD code

In order to run the MD code, you need to call `run_NVE()` which takes six compulsory arguments: coordinates, velocities, box size, # steps, # atoms, integration step.

For example, the simplest script could look like this:

```
Ncells = 6          # Number of unit cells along each axis
lat_par = 1.7048    # Lattice parameter
L = lat_par*Ncells  # Size of the simulation box
N = 4*Ncells**3     # Number of atoms in the simulation box
nsteps = 200        # Number of steps
dt = 0.003          # Integration step

# Generate fcc structure
pos, vel = crystal(Ncells, lat_par)

# Perform simulation and collect the output into a dictionary
output = run_NVE(pos, vel, L, nsteps, N, dt)
```

Files `Step*.py` contain a minimal setup needed to complete each step. In order to perform one step at a time make sure to comment the corresponding part of the code (steps are separated by descriptive comments).

To run MD with output on the screen: `python Step1.py`

To run MD with output on a file: `python Step1.py > testrun.out`

To run MD with output on both file and screen: `python Step1.py | tee testrun.out`

The first part of `Step1.py` will perform a constant energy calculation (NVE ensemble) with 200 steps (using a time step of 0.003), continuing from `sample10.dat` previously generated (or created by crystal), and writing on `sample11.dat` at the end.

On standard output (or inside `testrun.out`) you will find some important quantities monitored at each time step, such as kinetic and potential energies.

3. Compute velocities

In order to bring the sample close to the desired temperature (through constant velocity rescaling), we first need to compute the velocities for the atomic configuration generated with crystal. A small number of time steps (here, 200) is sufficient for this purpose.

The input file is stored as `md_start.in`:

```
nsteps = 200          # Number of steps
dt = 0.003            # Integration step

# Read crystal shape, positions and velocities from a file
N, L, pos, vel = read_pos_vel('sample10.dat')

# Perform simulation and collect the output into a dictionary
output = run_NVE(pos, vel, L, nsteps, N, dt)
```

For this tutorial, we will adopt an integration time step corresponding approximately to that used in [1] for liquid Ar (10^{-14} sec., see notes for the conversion to L.J. units). Among other things, you will be asked to check how your results depend on the time step: the value needed to ensure conservation of energy to a good extent depends on the temperature and on the particle density.

4. Change T and equilibrate

Now we are ready to apply the constant velocity rescaling to our sample: at each time step the velocities will be scaled in order to bring the instantaneous temperature of the system to the desired value ($T = 94.4$K, which corresponds to about 0.7867 in L.J. units for Ar).

The input is:

```
nsteps = 200
dt = 0.0046
T = 0.7867            # requested temperature

# Change T
output = run_NVE(output['pos'], output['vel'], L, nsteps, N, dt, T)

# Plot temperature vs step
plt.plot(output['nsteps'],output['EnKin']*2/3)
plt.show()
```

T is an optional argument of the function `run_NVE`, which default value is `None`. When T is greater than or equal to 0, the code will run a run at constant temperature. Notice that is NOT a constant energy dynamics, hence we are not sampling the NVE ensemble during this run (nor the NVT ensemble, see Task3 for NVT molecular dynamics). Since we are interested in the equilibrium properties (in the thermodynamics sense) of the system, no data should be collected in this kind of run, however you can see how the temperature changes during the run by plotting it against the step.

Before starting to collect data, we need to equilibrate the sample with a short run of regular NVE dynamics.

The input will be as follows:

```
nsteps = 800
dt = 0.0046

# Equilibrate
output = run_NVE(output['pos'], output['vel'], L, nsteps, N, dt)

# Plot total energy vs step
```

```
plt.plot(output['nsteps'],output['EnKin']+output['EnPot'])
plt.show()
```

By plotting the total energy, as a function of time you can check that E_{tot} is actually conserved (to a good approximation) in this kind of dynamics (and compare with what happens to E_{tot} in the constant velocity rescaling run). You can verify that the conservation of energy becomes more strict as the time step 'deltat' is reduced. In general, the other quantities display much larger fluctuations, instead. Notice that the average temperature might not be equal or not even close to the target temperature, since in the NVE dynamics is not possible to fix this variable (sometimes this makes also difficult to compare different MD simulations).

Step 2: Sample static properties

1. Compute $g(r)$ and $S(k)$ (through F.T.)

From the previously equilibrated atomic sample (which should be now stored in `sampleT94.4.dat`) you can start a MD run in which you do a sampling of some physical properties. We will first focus on some static properties, namely the radial pair correlation function $g(r)$ and the structure factor $S(k)$. The latter can be obtained in two modes, either directly by sampling the Fourier transform (FT) of the number density, or, in the case of an isotropic system, as the FT of the pair correlation function (see notes and Allen-Tildesly, ch. 2.6). In this subtask you will proceed through the second way.

The code can be used to perform a MD run of 2000 steps and evaluate the $g(r)$ at every step. The quantity is then averaged over all these samplings.

The code for $g(r)$ and $S(k)$ (through FT of $g(r)$) sampling is stored in `Step2.py`:

```
nsteps = 2000
dt = 0.0046
N, L, pos, vel = read_pos_vel('sampleT94.4.dat')

# Run MD simulation
output = run_NVE(pos, vel, L, nsteps, N, dt)

# Plot g(r)
plt.plot(output['gofr']['r'],output['gofr']['g'])
plt.show()

# Plot S(k)
plt.plot(output['sofk']['k'],output['sofk']['s'])
plt.show()
```

TO DO:

1. Measure position of the peaks (both in $g(r)$ and in $S(k)$) and compare to those reported by Rahman). Try to explain the other features you see.
2. Study the behaviour of these two quantities as a function of the equilibration temperature and of the density. For the former, you need to go through the steps seen before in order to bring the system close to the new temperature and equilibrate. For the latter you have either to generate a new sample with `cyrstal()`.

3. You may try a simulation with a larger number of atoms in order to extend the maximum radius allowed for $g(r)$, which is here limited to half of the box size (see next lectures for other methods to extend this limit). Be aware that when the number of atoms gets larger than a few thousands the code will become quite slow (due to the $O(N^2)$) operations. In order to overcome this you may have to use another version of the code which uses Verlet neighbor lists (at least for the dynamical evolution part).

4. When dealing with short range interactions (such as the LJ pair potential), the potential is approximated by truncating and setting it to a fixed value for interparticle distances beyond a certain cutoff radius (called `r_cutoff` in the code). By changing `r_cutoff` from its default value (2.5), you can check if and how this approximation affects the structural properties.

Step 3: Sample dynamical properties

Compute MSD and VACF

Now we move to the study of a dynamical quantity which is easily accessible through MD simulations: the diffusion coefficient. As you have learned during the class, this quantity can be computed from the mean square displacement (MSD) of the atomic positions through Einstein's relation, or from the integral of the velocity autocorrelation function (VACF).

Important remarks

1. ensemble average needs either to average on different time origins in the same run (not implemented) or to average (at same times) the quantity obtained as a function of time in several different runs (of same length). Smaller systems (small N) are subject to larger statistical fluctuations, therefore the deviation from the ensamble average may be quite large if a single realization is used to estimate the diffusion coefficient.

2. since we are sampling a dynamical quantity, the accuracy in the description of the particle trajectories plays more important role here. Be careful on the choice of `dt`; the value used so far may not be sufficiently small.

`Step3.py` adopts the second method for computing the ensemble average of the dynamical quantities, i.e. the MSD and VACF are calculated as an average over a number of simulations defined by `Nruns`.