TIF320
Computational materials and molecular physics

Assignment 3

Date: 2023-02-17

Students:
Afonso Sequeira Azenha
Irini Chrisovalantou Touloupi

Task 1

We have been given an AIMD trajectory of water (24 molecules), from which it is clear that the temperature fluctuates. Nevertheless, thermal equilibrium appears to be reached after about 1 ps (= 2000 time-steps $\times$ 0.5 fs). This can be seen from the fact that, after this time, the temperature fluctuates more or less around a central value of 350 K. To illustrate this, below, we present the graph obtained from plotting the temperature values, provided in the `cluster24.log` file, against time (Fig. 1). Additionally, we also used `ase gui cluster24.traj`, from where we could extract a graph of the variation of the kinetic energy, from which the same conclusions can be drawn (Fig. 2).

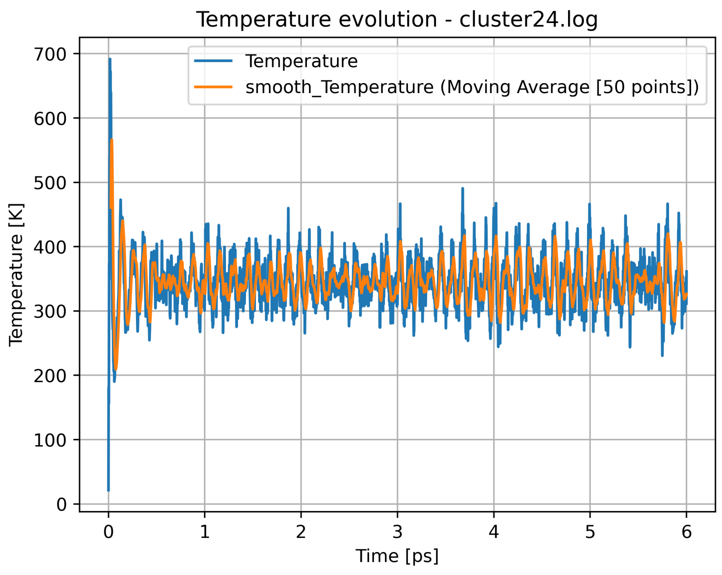

Figure 1: Temperature evolution graph. Obtained from the provided `cluster24.log` file. Moving average functions as a low-pass filter (to smooth out the signal).

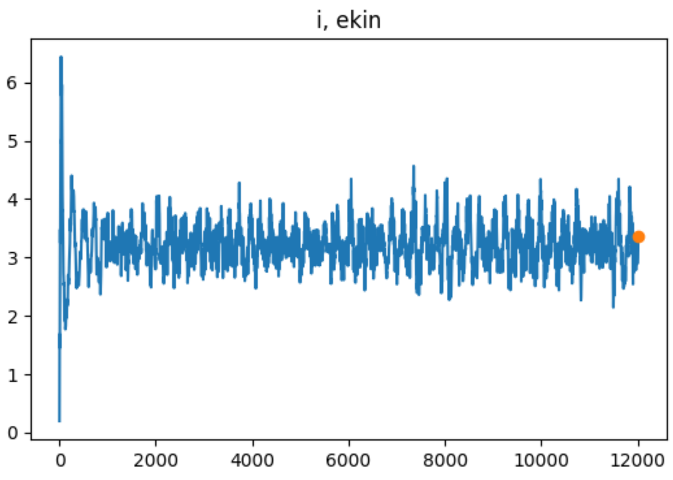

Figure 2: Temperature evolution graph. Obtained from the provided `cluster24.traj` file, using `ase gui cluster24.traj`. Labels: y axis - Kinetic energy, in eV; x axis - Snapshot number.

Using this information, we selected `'Snapshot_12000'` (from `cluster24.traj`) as our equilibrium Snapshot, and introduced a Na atom using `ase gui`'s drop-down menu. The obtained Snapshot will be included in our submission (`'Task1_Snapshot_12000.xyz'`). Nevertheless, we present a picture below.

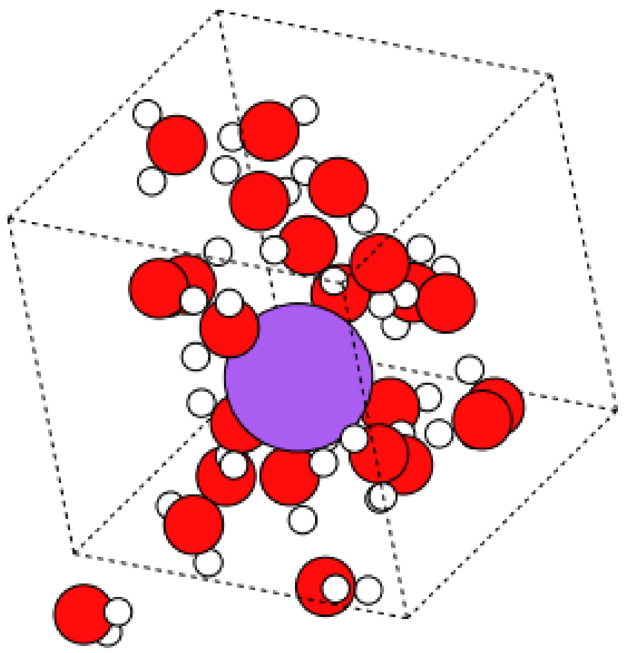

Figure 3: Utilized Snapshot, with added Sodium atom in the "middle".

It is important to note that the ion should not be placed too close to a water molecule, because, if so, the system could "explode". By placing the Na^+ ion too close to a water molecule, the potential becomes extraordinarily large, such that, by running the simulation, the ion might be shot out of the system. This would be the extreme case, in which the ion is placed exceptionally close to another water molecule. Under a not so drastic light, we could have the initial forces, in the first few time-steps, be considerably big, but not big enough to force this ejection. Under these conditions, the Na^+ ion would be strongly repelled, initially, leading to an increasing temperature of the system. However, since we have a thermostat in place, this would not last too long, since (kinetic) energy would be continuously pumped out of the system (by such thermostat), so as to achieve the desired (lower) target temperature. In one of our initial simulations, this was verified. We had put the Na^+ ion a bit too close to a water molecule, which resulted in the ion being strongly repelled, intially. However, due to the thermostat's influence, it entered an oscillatory motion, of decreasing amplitude, towards the imposed temperature value.

The coice of an appropriate timestep is limited by the oscillations of the molecule at hand. In order to have the best description possible, we need to be able to capture the fastest oscillations (light nuclei oscillate fast, e.g.). Below, we present a graph of the Attenuated Total Reflectance (ATR) *vs.* wavenumber, for water, which helped us decide on the value of dt.

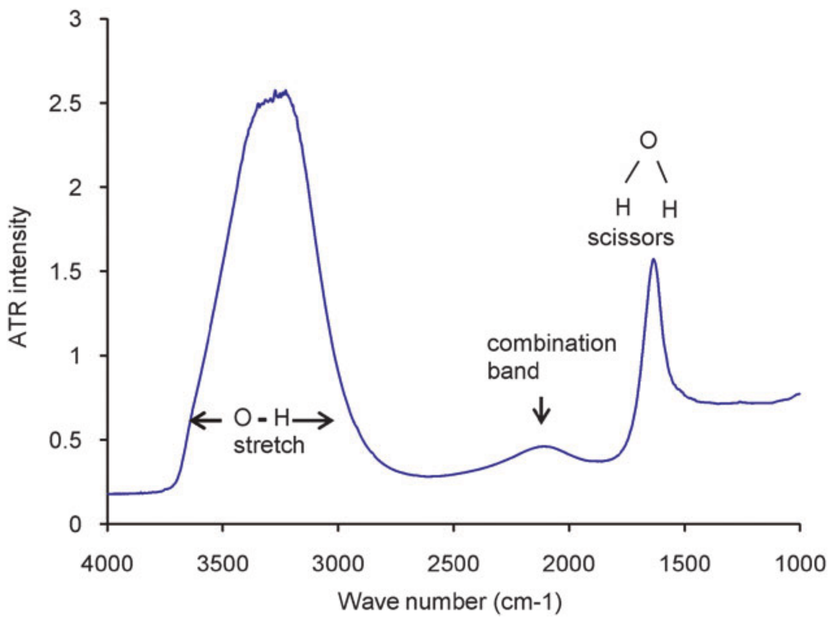

Figure 4: ATR-IR spectrum of water on ZnSe IRE. Taken from Ref. [3] (We used this graph in our lectures).

If we consider that 20-40 time-steps are required to accurately integrate an oscillation, the following calculations follow (keep in mind that 1 THz $\approx$ 33.36 cm^{-1}):

1. First, we pick the highest frequency oscillation (peak), which corresponds to the labelled O-H stretch. This corresponds to $\sim$ 3250 $cm^{-1} \approx$ 97.422 THz;

2. This, then, corresponds to (1/97.422) ps = 10.309 fs;

3. Finally, if we take the lower limit of 20 required time-steps (to capture the oscillation), we arrive at (10.309/20) fs = 0.515 fs. In our simulations, we use $dt = 0.5$ fs.

In order for the temperature to remain constant, there are various implementations that describe a coupling of the system to a heat bath (Canonical Ensemble - NVT). In our simulation, the Nosé-Hoover thermostat is used. Here, an extra term for the coupling to the heat bath is added to the Hamiltonian (initially, an additional coordinate s, with an effective mass Q, is added to the Lagrangian). This term has one degree of freedom and can be understood as a friction term that keeps the temperature constant by varying the value of the friction coefficient.

However, the Nosé-Hoover thermostat is not actually ergodic. "Ergodic" means that the ensemble average corresponds to the time average of the trajectory. This is important, since in (AI)MD simulations it is (much) easier to follow the system over a longer period of time, than to explicitly determine all of the possible states that it can assume. So, using the concept of ergodicity is important to get information on averages and dynamical processes.

Calculating the virtual time average, using Nosé's approach, is not the same as the real time average, since the real time steps are not equidistant. This problem of ergodicity can be solved by relating these two variables, using an integral over the virtual time to get the real time (see Ref. [1]). The Nosé-Hoover thermostat is not sufficiently chaotic in phase-space, being trapped in a subspace of it. In practice, this ergodicity problem is solved by considering what are called Nosé-Hoover chains.

This is, essentially, the implementation of "thermostatting" the thermostat variable, Q, which, effectively, allows us to probe a greater region of phase space (thus solving the ergodicity problem).

In DFT, forces can be computed through Hellmann-Feynman's theorem, which allows us to relate the obtained electron density (from DFT) to the forces acting on the nuclei. This theorem states that these forces can be readily calculated from the Coulomb interaction between the nuclei and this unperturbed electron density:

$$\mathbf{F}_n = -\frac{\partial E_0}{\partial \mathbf{R}_n} = -\left\langle \Psi_0 \left| \frac{\partial H}{\partial \mathbf{R}_n} \right| \Psi_0 \right\rangle = \int d^3 r n(r) \frac{Z_n \left(\mathbf{r}_i - \mathbf{R}_n\right)}{\left|\mathbf{r}_i - \mathbf{R}_n\right|^3} \tag{1}$$

(This represents the force acting on nucleus n.)

To calculate the forces, instead of DFT, one could use some sort of potential to model the interactions between atoms. Empirical interatomic potentials, based on either reactive (Pair potentials, such as Lennard-Jones, e.g.) or non-reactive force-fields (Harmonic, e.g.); Or, even, machine learning potentials could be used for this. Note that these are classical descriptions of fundamentally quantum phenomena. Thus, we cannot expect the results to be exact. Since these are, mostly, empirical potentials, one must be really careful in generalizing them. They might yield good results for some systems, while being outstandingly inaccurate for others. This stands in opposition to the DFT calculations, based on the Coulomb potential, that are, in theory, extendable to any possible system, although the calculations might become too complex, for too big systems. These classical descriptions allow for faster and lighter simulations, which is one of their main advantages, when compared to DFT.

Task 2

In this task, we compute the Radial Distribution Function (RDF), $g_{\alpha\beta}\ (r)$, between the Sodium ion and the water molecules. Since we want to look at the interaction with each "whole" water molecule, but not with all the individual atoms (there's lots of Hydrogen), only the Oxygen atoms of each water molecule and the Sodium ion are taken into account. The RDF we need to calculate has the following form:

$$g_{\alpha\beta}(r) = \frac{dn_r}{4\pi r^2 dr \rho}. \tag{2}$$

The variables we need to determine for this are dn_r and ρ. For dn_r, first, the distances between the Sodium ion and each Oxygen atom are determined.
For that, the `ase.Atoms.get distances` method is used, with the `kwarg`: `mic = True`, so as to indicate to ASE that we are using periodic boundary conditions. This is done for every Snapshot of the trajectory. Calculating, then, a histogram with all the obtained distances (of all the Snapshots) leads to the distribution of dn_r. This is then normalized by the number of Snapshots considered, such that the total number of Oxygen atoms (or, equivalently, water molecules) is still equal to 24. Note that we had to define a maximum distance from the Sodium ion, r_{max}, in order to compute the histogram. When choosing r_{max}, one needs to keep in mind that:

1. r_{max} will be used to compute the numerical density, ρ;
2. r_{max} cannot be too big, due to the finite size of the simulation cell (+ Periodic boundary conditions). If it is too big, then the sphereical shell, in which we shall

count the number of atoms inside, might extend to regions outside of the cell. This is a problem, since we know there are no atoms in these regions. However, this volume will still be considered in the normalizations that will follow, which introduces errors in our results, thus degrading their quality. For this reason, we maintain r_{max} = *(Length of Simulation Cell)*/2 ≈ 4.478, since the biggest possible sphere that can be fit in the Simulation Cell has a radius of half the cell's length. Note that, due to the utilization of periodic boundary conditions, one can always re-center the Simulation Cell around each individual atom, one at a time. That's why we can always have $r_{max} \approx 4.478$, even if the Na^+ ion is not perfectly centered in the middle of the cell. (This concept will re-appear later on, in Task 3, for the Oxygen RDF). Ideally, though, we could attempt to further develop the code, such that it detects these incidents and correctly computes the relevant volumes (by removing spherical caps off the spheres, when they go overboard). This would allow us to go further beyond, to higher values of r_{max}. However, this was not implemented.

The value of ρ (number density) is obtained from the sphere of radius r_{max}, centered around the Sodium ion. The number of atoms inside this sphere is counted and, subsequently, divided by its volume, yielding ρ. In practice, $\rho = \frac{\textit{Number of Oxygen atoms}}{(4/3)\cdot\pi r_{max}^3}$. Below, we present the parts of the code in which all of this is implemented.

```python
# Definitions:
n_bins = 100
r_max  = Cell[0, 0]/2      # Periodic boundary conditions means we can
     re-center ('the cell') around each individual atom. So, r_max =
    Cell_x/2.
max_d = [] # Checking what is the biggest distance that is verified.
global_count = np.zeros(n_bins)

for snapshot in trajs:
    atoms = snapshot[:24]      # Selecting the 24 'O' atoms.
    atoms.append(snapshot[-1]) # Adding the 'Na' atom.

    dist = atoms.get_all_distances(mic = True)[-1][:-1] # [:-1] to get
     rid off the Na distance (0).
    max_d.append(sorted(dist)[-1])

    counts, bins  = np.histogram(dist, bins = np.linspace(0, r_max,
    n_bins + 1, endpoint = True))

    global_count += counts

# Averaging over all the snapshots:
global_count /= snap_n
```

Listing 1: Building histogram from all the Snapshots.

```python
# Number density:
Vol_Max = (4/3)*np.pi*(r_max**3)
rho = sum(global_count)/Vol_Max

# RDF Computation:
for bin_count, r, i in zip(global_count, bins, ind):
    if (bin_count > 0):
        Volume = 4*np.pi*(r**2)*dr
        final_count[i] = bin_count/(Volume*rho) # This is g.
```

Listing 2: Computation of $g_{\alpha\beta}(r)$.

Using this code, we were able to obtain $g_{NaO}(r)$. The results are presented below.

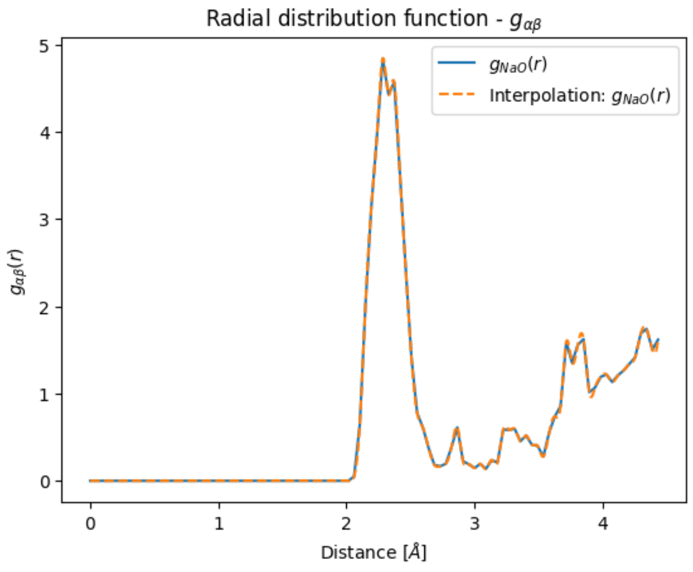

Figure 5: Radial distribution function, $g_{NaO}(r)$. Computed using our AIMD simulation.

The same $g_{NaO}(r)$ was also computed for the reference `.traj` file, provided in the TIF320 `git` repository.

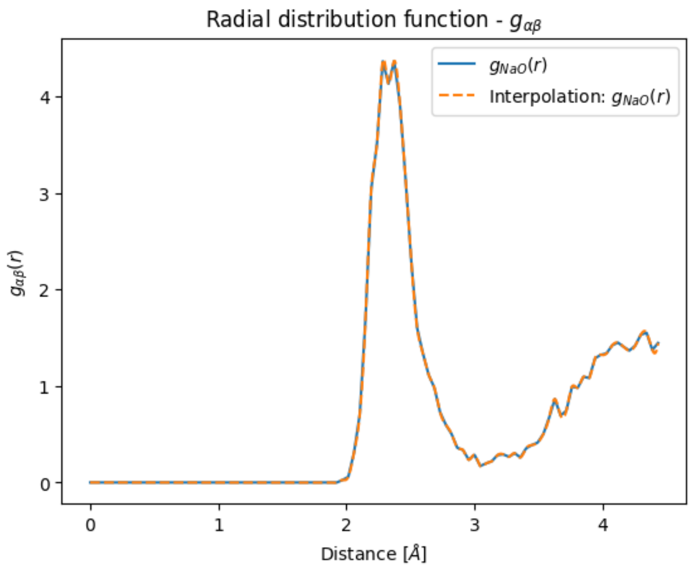

Figure 6: Radial distribution function, $g_{NaO}(r)$. Computed using the provided `NaCluster24.traj` file.

Additionally, we opted to interpolate (orange line) the obtained results, before integrating to obtain the first solvation shell. Initially, we had considered 100 bins, from $r = 0$ to $r = r_{max} \approx 4.478$. Interpolating means that we have access to a greater number of points, which in turn means that we should be able to more accurately determine the wanted minimum. Alternatively, we could have just considered a greater number of bins in the histogram (Although we should keep in mind that overbinning can be a problem.)

Next, we want to calculate the first solvation shell of the Na^+ ion. It is defined as follows:

$$n(r') = 4\pi\rho \int_0^{r'} g_{NaO}(r) r^2 dr.$$

This corresponds to the integral of the RDF ($\times\rho$) (between the ion and the water molecules) up to its first minimum, r'. We found $r' = 2.710$ Å, for our AIMD simulation (Fig. 5), and $r' = 3.056$ Å for the reference `NaCluster24.traj` file (Fig. 6). The relevant code snippet is presented below. Note that, when computing the RDF's, we opted to discard the first 1000 Snapshots. This guarentees that equilibrium (in Temperature) is reached (Figs. 7 and 10), by neglecting any possible transients at the start.

```python
# Integrand:
func = lambda r: 4*np.pi*r**2 * rho * Interpolator(r)

# First solvation shell:
a = 0; b = fmin(Interpolator, 2.5)[0] # b - Minimum value (r).
print(f'Found minimum: r = {b}.')
print(f'1st solvation shell = {quad(func, a, b, limit = 150)}.') #
    Integrating until the obtained r value.
```

Listing 3: Computation of the first solvation shell.

Calculating this, with our own trajectory, leads to a value of 3.463, which is quite far away from the experimental value of 5. This number tells us that this ion is (should be) surrounded by 5 water molecules (on average), while completely solvated in water. Thus, it appears that our short simulation was not enough to capture the ion's (complete) solvation in water. In an attempt to diagnose why this was the case, we decided to plot the variation of the temperature, along the simulation time. The result is presented below. It appears that the temperature was, indeed, in equilibrium, so that was not the problem. Perhaps, the simulation simply needed more time to fully capture the dyanimcs of this system.

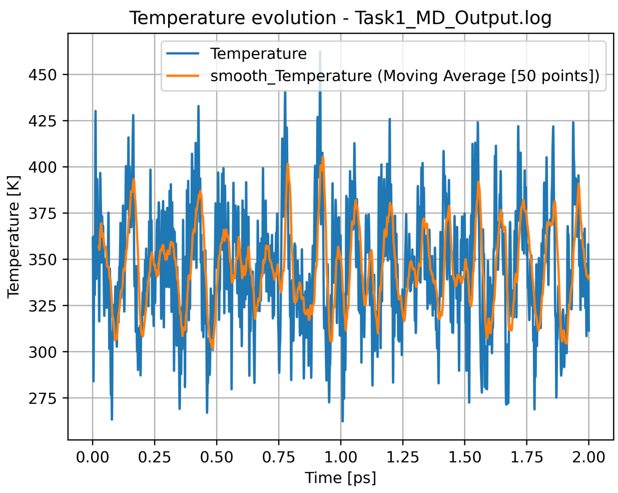

Figure 7: Temperature evolution graph. Obtained from our AIMD simulation's `.log` file.

To further investigate this, we also plotted the variation of the total energy of the system. The obtained results are presented below.

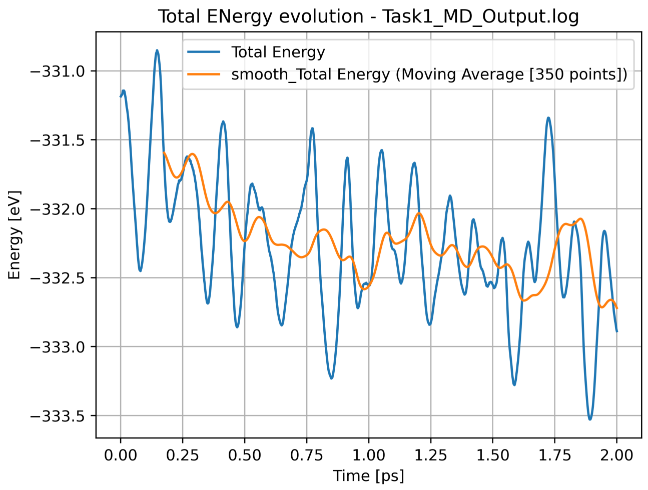

Figure 8: Total energy's evolution graph. Obtained from our AIMD simulation's `.log` file.

From this plot, it appears that our energy value is still varying. This could indicate the occurrence of some chemical process, such as a chemical reaction (which shouldn't be the case), or, even, a solvation process! The fact that the energy is still changing could mean that the solvation has not yet "ended". We arrive at the simple conclusion that we needed to simulate more time, in order to fully capture the dynamics of this solvation process.

Additionally, we'd like to comment on the orientation of the water molecules around the solvated ion. Below, we present the last Snapshot from our simulation, as an illustration of what we're about to discuss.

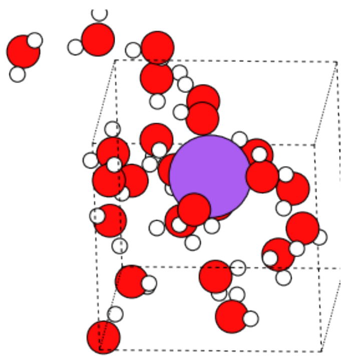

Figure 9: Last Snapshot of our AIMD simulation. Note the orientation of the H_2O molecules, around the Na ion.

As can be seen, the molecules that make up the first solvation shell (in this Snapshot) all have their Oxygen atom turned towards the Sodium ion. Intuitively, this makes perfect sense, as we know there's extra negative charge around the Oxygen atom (higher electronegativity of O, relative to H), in the H_2O molecule (H_2O is a dipole!), which is attracted to the positive charge associated with the Na^+ ion. In addition, further away from the ion, it appears that we have lots of hydrogen bonds between water molecules. It can be seen (although not so clearly in this Figure) that the water molecules orient themselves such that a Hydrogen atom (positive part of the H_2O dipole) is always "pointing" towards an Oxygen atom (negative part of the H_2O dipole), which, again, makes perfect sense.

On the other hand, for the longer `.traj` file provided, the first solvation shell was determined to be 4.445, which is much better (Here, we also neglected the first 1000 Snapshots, when determining the RDF!). For comparison with the above case, we also plotted the temperature evolution graph in this case. Once again, we appear to have reached equilibrium in temperature.

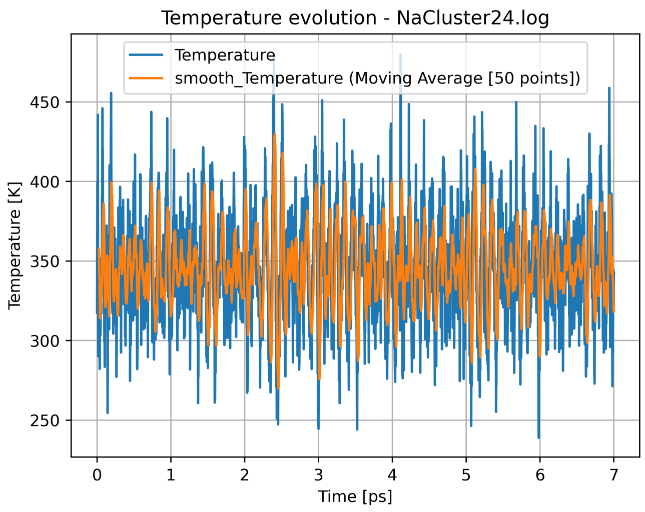

Figure 10: Temperature evolution graph. Obtained from the provided `NaCluster24.log` file.

Task 3

The different functionals used (in DFT) for computing the RDF between Oxygen atoms in a water AIMD simulation are presented in Figure 1 of the Assignment's question paper. Naturally, the best functional will be the one that gives results which are the closest to the experimental curves. So, we believe that the fuctional 'optB88-DRSLL' is best-suited for simulating water, since it is the closest to the experimental curve. The other functionals are either over or underestimating the maximum and/or the minimum of the RDF.

We now compute the RDF between Oxygen atoms, after thermal equilibrium, for the (Sodium-free) water trajectory we have been given. The essence of the code remains pretty much the same as in Task 2, except for two things:

1. Now, we do not take the Na^+ ion into account;
2. In addition, we now need to compute the RDF for each Oxygen atom (individually) in the simulation, so we can then average the results over all the Oxygen atoms. The relevant code snippet is presented below. For additional details, check the file `Task3.ipynb` provided in our submission (with the code).

```
# Definitions:
n_bins = 100
r_max  = Cell[0, 0]/2
max_d = []
global_count = np.zeros(n_bins)

```

```
for snapshot in trajs:
    temp_counts = np.zeros(n_bins)
    for i in range(24):
        atoms = snapshot[:24] # Selecting the 24 'O' atoms.

        dist = np.delete(atoms.get_all_distances(mic = True)[i], i) # Need np.delete to get rid of the 'self-distance'.
        max_d.append(sorted(dist)[-1]) # Checking what is the biggest distance that is verified.

        counts, bins  = np.histogram(dist, bins = np.linspace(0, r_max, n_bins + 1, endpoint = True))

        temp_counts += counts
    global_count += temp_counts/(24) # Number of Oxygen atoms!

# Averaging over all the snapshots:
global_count /= snap_n
print(f'Maximum verified distance = {sorted(max_d)[-1]}.')
```

Listing 4: Building histogram from all the Snapshots. The rest (computing $g_{OO}(r)$ from the histogram) is the same as before (in Task 2).

The following RDF was obtained.

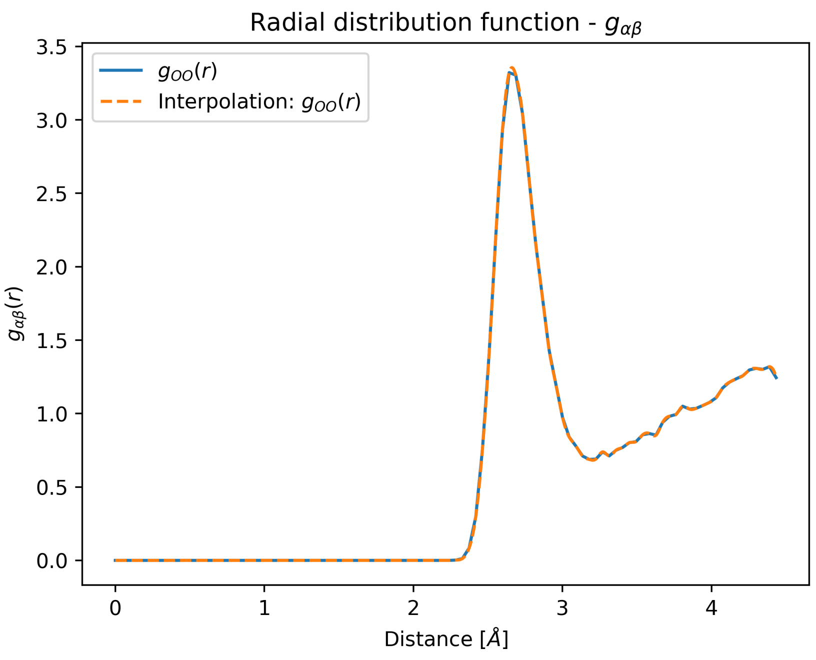

Figure 11: Radial distribution function, $g_{OO}(r)$. Computed using the provided `cluster24.traj` file.

Note: The first 2000 Snapshots were rejected (for computing the RDF), since the temperature had not yet reached equilibrium before then. To see this, check Figures 1 and 2.

In this case, we considered the dynamics of 24 water molecules, which yielded reasonable results, (qualitatively) comparable to the experimental curves provided, although it appears we seriously overestimate the first peak of g_{OO}. Our results seem to match the curve for 'PBE-D3' the most. This exchange-correlation functional is,

essentially, an extention of PBE to include Van der Waals interactions, so it makes sense that our calculations, performed using PBE, yield similar results.
Since it is obviously cheaper to run the simulation with less water molecules, it is now a question of whether a smaller simulation will give us anywhere near as good results. As stated above, the RDF is calculated using the distances between the Oxygen atoms. If we were to consider fewer molecules, then fewer distances would be included in the calculations. The RDF would then have a bigger statistical variance. As a result, this would lead to poorer statistics, which means the smaller cell might not necessarily capture the same physics. Additionally, if we consider a smaller cell size, with fewer water molecules, then, naturally, our RDF will appear to be skewed towards lower values of r. This way, we might end up losing some of the characteristic behaviour of our water molecules at greater distances. Besides, just 7 molecules might also not be enough to capture much more than the dynamics of the first neighbours (of a single H_2O molecule). These correspond to the first peak in Figure 11. In fact, if we compute the same integral that we did to determine the first solvation shell of Na^+ (in water), we get, in this case, 4.092 ($r' = 3.204$ Å), which further supports our claim that only 7 molecules won't be able to describe much more than the first neighbours of a water molecule. Thus, we might end up neglecting possibly important dynamics that could happen if we had more H_2O molecules in the simulation.

There are some theoretical and numerical approximations made in the utilized variant of DFT. One theoretical approximation is, obviously, the chosen exchange-correlation functional. Furthermore, we are using LCAO, which is sub-optimal most of the time. The fact that we consider an incomplete basis set is exactly what makes this an approximation. In order to improve on this, we could opt to use, for instance, a rather dense finite-differences grid, although this would surely be more computationally expensive. We also use periodic boundary conditions as another practical approximation. Additionally, we should be able to improve on these approximations by using a larger cell size, with more molecules. Obviously, this would be more expensive, but could also prove to be more accurate. Ultimately, one could try to find better XC-Functionals (although this is quite farfetched - it's an entire field on its own!).

References

[1] Thijssen, J.M. (2013) Computational physics. Cambridge: Cambridge University Press. Chapter 8.5

[2] ASE Molecular dynamics. Available at: `https://wiki.fysik.dtu.dk/ase/ase/md.html#constant-nvt-simulations\-the-canonical-ensemble`

[3] Mojet, Barbara & Ebbesen, Sune Lefferts, Leon. (2010). ChemInform Abstract: Light at the Interface: The Potential of Attenuated Total Reflection Infrared Spectroscopy for Understanding Heterogeneous Catalysis in Water. Chemical Society reviews. 39. 4643-55. 10.1039/c0cs00014k.