Assignment 3 – Computational Materials and Molecular Physics
First hand-in

Eric Lindgren – CID: ericlin

February 20, 2020

1 Task 1: Running the calculations

Studying the given trajectory of only the water molecules in `cluster25.traj` using ASE GUI I conclude that equilibrium has been reached after around 8000 time steps, since the energy of the system fluctuates randomly around some mean value seemingly independent of the initial condition. From the logifle `cluster24.txt` the time step is determined to be $\Delta t = 0.5\,\text{fs}$. To be safe, I use the configuration at 10000 time steps as my equilibrated starting configuration, corresponding to a simulation time of 5 ps, and insert a sodium atom into the system. This configuration was then saved to the file `thermalizedConfiguration.xyz`. The modified snapshot was then used in the script `task1.py`, see appendix A.1 for specific details, as the starting configuration for an AIMD simulation. The total charge of the system was set to +1 due to the sole Na^+ ion.

Choosing the correct time step for a MD simulation is a balance between computational ease, which is increased with a larger time step, and accuracy of the obtained trajectories which is increased with a smaller time step [1]. According to the lecture [2], a common rule of thumb is to select the time step such that the quickest oscillations in the system can be sampled at least 20 times per period. In our system which is mostly of water, the fastest oscillations are the O–H stretching of the water molecule with a frequency of $\sim 100\,\text{THz}$. If a signal with this frequency is to be sampled 20 times per full period, a sampling frequency of

$$20\Delta t = \frac{1}{100 \cdot 10^{12}\,\text{Hz}} \rightarrow \Delta t = \frac{1}{2 \cdot 10^{15}\,\text{Hz}} = 0.5\,\text{fs} \tag{1}$$

is required. Thus I settled for simulating the system with a $\Delta t = 0.5\,\text{fs}$.

If a system is to be kept at constant temperature, then it must be coupled to a heat-bath. This is modelled in Nosé–Hoover dynamics by adding a term corresponding to the coupling to a heat bath in the Hamiltonian for the system [3].

DFT plays the role of computing the inter-particle potential in the system and from there the forces between the particles, which is then used to classically update their positions in the AIMD simulation. The classical update can for instance be done using the Velocity Verlet algorithm [2]. That the positions are updated classically doesn't generally impact the results from the simulation that much; all of the relevant quantum physics is hidden in the calculation of the forces. Thus, one could use some other method for calculating the potential in the system and from there the forces. However, DFT is a relatively efficient method for handling systems of many particles quantum mechanically, especially when compared to for example Hartree or Hartree–Fock based methods which quickly becomes compuationally intractible. However, if a quantum modelling of the interaction of the particles in the system is unnecessary, one could use classical methods to estimate the forces between the particles instead of DFT, which would probably be much less computationally intensive.

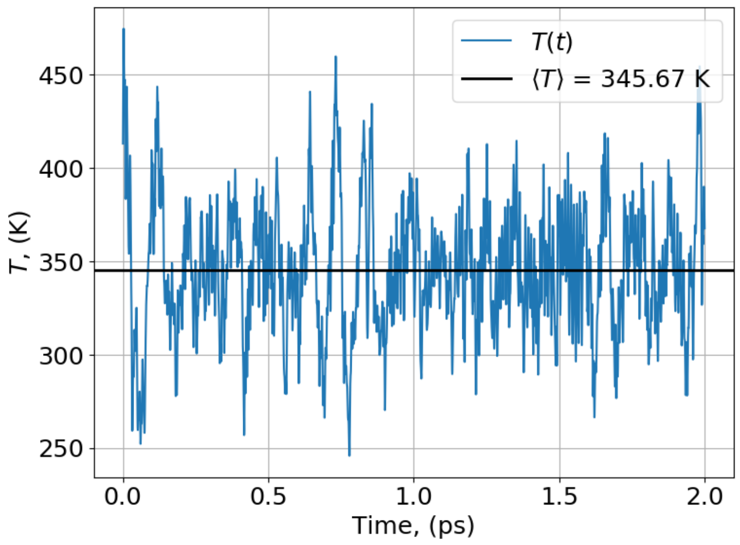

Figure 1: Temperature trajectory of the system during a simulation time of 2 ps. Note that the system seems to be equilibrated almost immediately, but to be safe I've chosen to deem the system as equilibrated after 0.5 ps.

The simulation on Hebbe with the settings described above, which can be studied more in detail in appendix A.1, took around 100 CPU-hours. The temperature of the system during the simulation time of 2 ps is given in figure 1. Note that the temperature oscillates with an amplitude of $\sim$ 100 K around the target temperature of 350 K, with an average temperature of $\langle T \rangle \simeq 346$ K. The fluctuations are expected from the use of a thermostat due to the limited size of our simulation cell [4]. Furthermore, we don't observe any large temperature spikes during the early parts of the simulation. This is contrary to what I expected; my hypothesis was that the insertion of the Na^+-ion would disturb the system greatly and that it would take some time for it to equilibrate. But the results point to the system being seemingly equilibrated almost from the start. This could be due to the careful placement of the Na^+-ion in a region that was devoid of any other particles, thus minimizing the changes to the system. However, to be safe I will consider the trajectory to be equilibrated after 0.5 ps, i.e. after 1000 time steps, in the following task.

2 Task 2

The radial distribution function (RDF) $g(r)$ contains information about the probability of finding a particle at a distance r from another particle, and is defined as:

$$g(r) = \frac{\mathrm{d}n_r}{4\pi r^2 \mathrm{d}r\rho} \tag{2}$$

where $\mathrm{d}n_r$ is the number of particles in an infinitesimally thin shell of length dr and at a distance r [5]. In the denominator of equation (2) we find an expression which normalizes the RDF; $4\pi r^2 dr\rho$ is the number of particles that one would find in a spherical shell at radius r if the particles where uniformally distributed according to the number density $\rho = N/V$, i.e. that the particles were uncorrelated and could be found anywhere in the available volume V. As $r \to \infty$ we would thus expect that the RDF approaches 1, i.e. that the particles become uncorrelated.

In the partial RDF $g_{\alpha\beta}(r)$, one only sums over the relevant subspecies of particles in the system [4]. In our case the only relevant particles is the Na^+-ion and the oxygen atoms. This is due to the Na^+-ion being positively charged and the oxygen atom in the water molecule being slightly negatively charge [6], and hence they will attract each other. The hydrogen atoms in the water molecules are slightly positively charged, which combined with the oxygen atoms being negative means that the water molecules will surround the Na^+-ion with their oxygen atoms inwards. Thus we only really need to consider the oxygen atoms when analysing the solvation of an Na^+-ion in water. Our partial RDF will then be $g_{\mathrm{Na^+O}}$.

I computed this RDF by going through the trajectory obtained in task 1 and calculating the distances between the Na^+-ion and all the oxygen atoms for each time step, under the condition of the minimum image convention in order to implement the periodic boundary conditions. The distances where then binned into a histogram, which was in turn divided by the number of snapshots in the trajectory and the normalization factor $4\pi r^2 dr\rho$. The first solvation shell, or the coordination number of the Na^+-ion, can roughly be thought of as the average number of water molecules that are closest to the ion. Denoting this quantity N, we can obtain it by using equation (2)

$$N = \int \mathrm{d}n = \int_0^{r_{min}} \frac{\partial d}{\partial dr} n \mathrm{d}r = 4\pi\rho \int_0^{r_{min}} r^2 g_{\mathrm{Na^+O}}(r)\mathrm{d}r \tag{3}$$

where r_{min} corresponds to the first minimum of the RDF $g_{\mathrm{Na^+O}}$. The minimum corresponds to the first "gap" in the distribution of particles around the Na^+-ion. For implementation-specific details, please see the code in appendix A.2.

The same procedure was applied for extracting the partial RDF and the coordination number for the Na^+-ion from the provided trajectory `NaCluster24.traj`, which was 7 ps long. I considered the trajectory equilibrated after 8000 time steps (4 ps), which I estimated from studying the energy of the trajectory using the ASE GUI.

The obtained RDF:s for the equilibrated parts of my and the given trajectories can be seen in figure 2. The corresponding coordination numbers where 4.88 and 4.45 respectively. Note that both the RDF and the coordination number are similar for both trajectories. The slight difference between the coordination numbers could indicate that the short simulation of 2 ps is not enough to fully capture the solvation of the Na^+-ion, but that it is a good approximation. The coordination number tells us how many oxygen atoms (and thus water molecules) on average surround the Na^+-ion [6]. Thus, my obtained coordination number means that on average 4.88 water molecules surrounded the Na^+-ion during the simulation.

We also observe that the obtained values for the coordination number for both trajectories are in line with the experimental value of ~ 5. The small difference may be due to the choice of XC-functional when simulating this system. We used the PBE functional, but for water hybrid functionals often performs better [6]. See the beginning of section 3 for a deeper discussion on this topic.

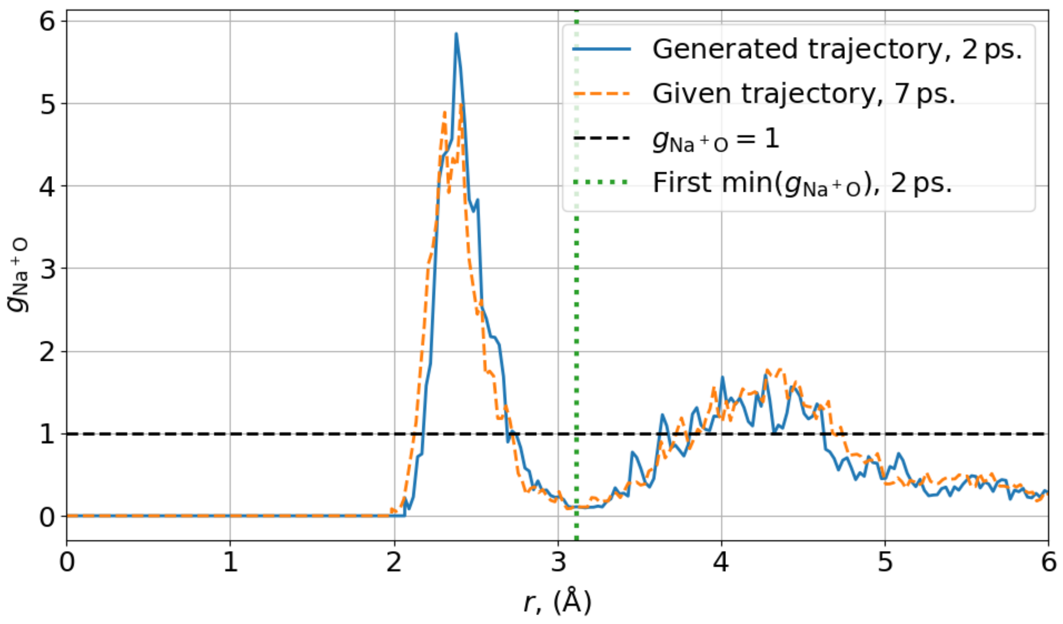

Figure 2: The partial RDF g_{Na^+O} for the equilibrated parts of my simulated trajectory (2 ps) as well as for the given trajectory (7 ps). The first minimum of the RDF from my simulated trajectory is also given. Note the qualitative similarity between the two RDF:s.

3 Task 3

From studying the radial distribution functions in Figure 1 in the problem description [4], I estimate that the trajectory given by the functional BLYP-D3 most closely follows the experimental results, and hence this would be my choice for simulating water out of the functionals available in the figure. BLYP-D3 is the BLYP functional with an empirical correction (D3) [7]. An interesting thing I found from googling the different functionals was that all of them seem to be pure DFT functionals; there is no hybrid functional. Hybrid functionals mixes the "classical" density approximating functionals from DFT with some parts Hartree-Fock to better handle the exchange in the system [8]. The Pauli exchange is perfectly modelled, by construction, in Hartree-Fock. According to the lecture given on this topic in the course, one often needs hybrid functionals in order to get the correct properties for water [6]. So even if the functionals in Figure 1 closely represent the experimental results I would choose a hybrid functional.

We use almost the same script as in task 2 to compute the RDF for the water trajectory after equilibration, which in task 1 was determined to be after 10000 time steps (5 ps). One difference between the scripts for task 2 and task 3 is that in task 3 we compute the distance between all pairs of oxygen atoms to obtain better statistics. In task 2 we only had one Na^+-ion. Since there are 24 oxygen atoms in total, and in order to not double count the distances between them, the distance between the first oxygen atom and the remaining 23 are calculated. Then, the distance between the second oxygen atom and the remaining 22 atoms, and so forth. The last oxygen atom is ignored, it has already been compared to the other 23. This means that we compare each oxygen atom to on average

$$\frac{1}{N}\sum_{n=1}^{N} N - n = \{N = 24\} = \frac{276}{24} = 11.5 \tag{4}$$

other oxygen atoms. We thus need to divide the counts in the histogram for radial distances with 11.5 times the number of time steps that we we compute the histogram over, which in our case is 3000 time steps (1.5 ps), in order to get a properly normed RDF. See `task3.py` in appendix A.3 for implementation specific details. The obtained RDF is given in figure 3. By comparing the obtained RDF with the experimental results in Figure 1 in the problem description, we note that it qualitatively seems to follow the experimental results fairly well but that the amplitude is too large. My obtained RDF is very similar to the simulated RDF for the PBE-functional in Figure 1, which isn't so surprising since I've also used PBE. The discrepancy between my obtained RDF and the experimental results could hence indicate that the PBE functional doesn't describe the system correctly. Another difference between the experimental results and my results in figure 3 is that my RDF approaces 0 as r increases, whilst the experimental result approaces 1. This is due to the limited size of the cell that the system is simulated in with a cell side length of ~ 9 Å.

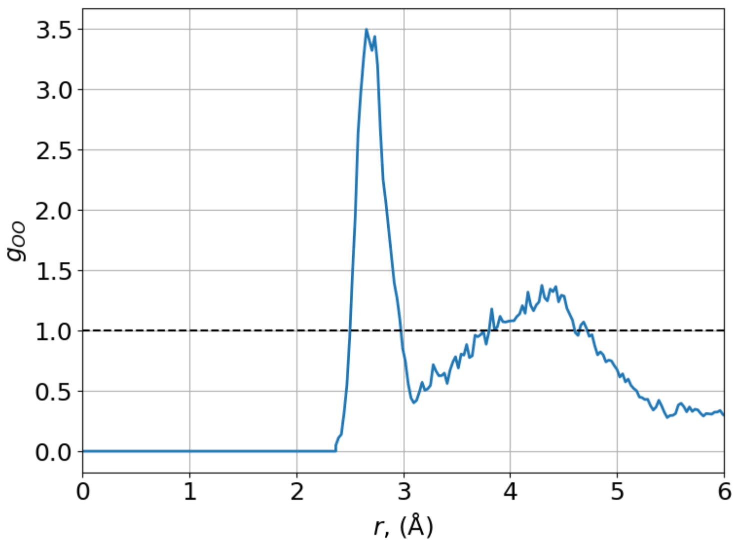

Figure 3: Partial radial distribution function g_{OO} between oxygen atoms in a cell consisting only of water.

If we where too reduce the number of water molecules in the system from 24 to 7, I would argue that we would probably not get the correct dynamics of the solvation of the Na^+-ion. The solvation number for the Na^+-ion as obtained in task 2 was $\sim$ 5, but there the attraction of the oxygen atoms and the Na^+-ion was counteracted by the slight interaction inbetween the water molecules. Having only 7 water molecules would probably not be sufficient to capture the interaction inbetween the water molecules, which I hypothesize would lead to an overestimation of the coordination number of the Na^+-ion. Instead, we would probably see that all 7 water molecules would surround the Na^+-ion. Hence in this particular case it would probably be detrimental to the simulation to lower the number of water molecules so far, even though it would be less computationally intensive.

References

[1] G. Wahnström. *Molecular Dynamics Lecture Notes*. Accessed: 202-02-05. 2019. URL: %5Curl%7Bhttps://chalmers.instructure.com/courses/7636/files/231506?module_item_id=42011%7D.

[2] Julia Wiktor. *Lecture 6 – AIMD*. Accessed: 2020-02-18. 2020. URL: %5Curl%7Bhttps://chalmers.instructure.com/courses/8949/files?preview=372395l%7D.

[3] ASE-developers. *Molecular dynamics – ASE documentation*. Accessed: 2020-02-05. 2017. URL: %5Curl%7Bhttps://wiki.fysik.dtu.dk/ase/ase/md.html%7D.

[4] Nicklas Osterbacka. *Home Assignment 3: Na+ solvation in water*. Accessed: 2020-02-18. 2020. URL: https://chalmers.instructure.com/courses/8949/files?preview=376258.

[5] Wikipedia. *Radial distribution function*. Accessed: 2020-02-18. 2019. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/Radial_distribution_function%7D.

[6] Julia Wiktor. *Lecture 8 – Non–crystalline*. Accessed: 2020-02-18. 2020. URL: %5Curl%7Bhttps://chalmers.instructure.com/courses/8949/files?preview=386746%7D.

[7] Xavier Assfeld. *Quantum Modeling of Complex Molecular Systems*. From Google Books. Accessed: 2020-02-18. Springer, 2015, p. 204.

[8] Wikipedia. *Hybrid functional*. Accessed: 2020-02-18. 2019. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/Hybrid_functional%7D.

A Python scripts

A.1 task1.py

Please run both scripts in the folder ”task1”.

```python
# Built-in packages
import time

# Third-party packages
import numpy as np

from ase import Atoms
from ase.io import read, write, Trajectory
from ase.visualize import view
from ase.units import fs, kB
from ase.md.npt import NPT
from ase.parallel import world

from gpaw import GPAW

# Load atoms object
a = read('thermalizedConfiguration.xyz')
a.wrap()
# view(a)  # DEBUG

# Sanity check - check distance between atoms
O_idx = [O.index for O in a if O.symbol=='O']
Na_idx = [Na.index for Na in a if Na.symbol=='Na']
distances = a.get_distances(Na_idx, indices=O_idx, mic=True)  # Enable minimum image ↩
    convention to check pbc
assert min(distances) >= 1  # Check if Na is far away from the oxygen molecules

# Define timestep, total simulation length and number of steps
dt = 0.5*fs
t_tot = 2000*fs  # 2 ps
N_steps = int(t_tot/dt)
if world.rank==0:
    print(f'------------    MD simulation with GPAW for {N_steps} steps    ↩
        ------------')
start = time.time()
calc = GPAW(mode='lcao',
            xc='PBE',
            basis='dzp',
            symmetry={'point_group': False},
            charge=1,
            h=0.2,
            txt='out_mdTask1.txt'
)

a.set_calculator(calc)

dyn = NPT(
    atoms = a,
    timestep = dt,
    temperature = 350*kB,
    externalstress = 0,
    ttime = 20*fs,
    pfactor = None,
    logfile = 'log_mdTask1.txt'
) # Using the Nos -Hoover thermostat

trajectory = Trajectory('mdTask1.traj', 'w', a)
dyn.attach(trajectory.write, interval=1) # Write state of the system to trajectory at↩
     every timestep
dyn.run(N_steps)
end = time.time()
if world.rank==0:
    print('-------- MD simulation finished in: ' + f'{(end-start):.2f} s --------'.↩
        rjust(34))
    print('----------------------------------------------------------------------')
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Set plot params
plt.rc('font', size=18)           # controls default text sizes
plt.rc('axes', titlesize=18)      # fontsize of the axes title
plt.rc('axes', labelsize=18)     # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)     # fontsize of the tick labels
plt.rc('ytick', labelsize=18)     # fontsize of the tick labels
plt.rc('legend', fontsize=18)     # legend fontsize

data = np.loadtxt('log_mdTask1.txt', skiprows=1)
avgT = np.mean(data[:,4])

fig, ax = plt.subplots(figsize=(8,6))
# data.plot(kind='line', x='Time', y='T', ax=ax)
ax.plot(data[:,0], data[:,4], linestyle='-', label=r'$T(t)$')
ax.axhline(avgT, c='k', linewidth=2, label=rf'$\left<T \right>$ = {avgT:.2f} K')
ax.set_xlabel('Time, (ps)')
ax.set_ylabel(r'$T$, (K)')
ax.legend(loc='upper right')
ax.grid()
plt.tight_layout()
plt.savefig('task1_T_traj.png')
plt.show()
```

A.2 `task2.py`

Please run in the folder ”task2”.

```python
# Built-in packages
import os.path

# External packages
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.signal import medfilt
from ase.io.trajectory import Trajectory
from tqdm import tqdm

# Set plot params
plt.rc('font', size=18)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=18)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)    # fontsize of the tick labels
plt.rc('ytick', labelsize=18)    # fontsize of the tick labels
plt.rc('legend', fontsize=18)    # legend fontsize

'''
Calculate the first ion solvation shell for my simulated trajectory from task 1 and
from the given trajectory in NaCluster24.traj.
The RDF is calculated from the position of the Na-ion (i.e. Na is at r=0) since we
are interested in the solvation shell of the ion.

The relevant species for the partial-RDF is the Na+ ion and the oxygen molecules,
due to the hydrogen atoms being bound to the oxygen. Thus the positions of the
hydrogen atoms needs to be integrated out.

The RDF is given as a histogram over the distances from the Na+ ion to the oxygen
atoms for all snapshots.
'''

def load_distances(file, trajectory):
    distances = [] # Holds all distances from Na+ for all snapshots
    if not os.path.exists(f'{file}_{len(trajectory)}_steps.npy'):
        print('---- Creating distances vector ----')
        for snapshot in tqdm(trajectory):
            # Get indices for Na+ ion and oxygen atoms
            O_idx = [O.index for O in snapshot if O.symbol=='O']
            Na_idx = [Na.index for Na in snapshot if Na.symbol=='Na']

            # Calculate their distances
            distances.extend(snapshot.get_distances(Na_idx, indices=O_idx, mic=True))↩
                  # Enable minimum image convention to check pbc
        distances = np.array(distances)
        print('---- Saving to ' + f'{file}_{len(trajectory)}_steps.npy ----'.rjust↩
            (20))
        np.save(f'{file}_{len(trajectory)}_steps.npy', distances)
    else:
        print('---- Load from ' + f'{file}_{len(trajectory)}_steps.npy ----'.rjust↩
            (30))
        distances = np.load(f'{file}_{len(trajectory)}_steps.npy')
    return distances


def generate_partial_RDF(distances, n_snapshots):
    n = 200  # Number of points
    # Get a plot of the RDF from its histogram - it is accurate for sufficiently small bins
    RDF, b = np.histogram(distances, bins=n)  # Return the bin edges and use the as r vector for normalization
    RDF = RDF.astype(float) / n_snapshots  # Get the average occupation in each box over all snapshots
    r = np.array([(b[i-1]+b[i])/2 for i in range(1, len(b))])  # Position is middle point of each bin
    dr = r[1]-r[0]  # The size of the spherical radial shell

    # Normalize the obtained distribution function RDF(r) with the number density times the volume of the spherical shell at radius (r)
    # This would have been the RDF had the particles been uncorrelated (on average an rho particles per unit shell).
    V = traj[0].get_volume()
    rho = 24/V  # Number density of the relevant species of the system - ( 24 O + 1 Na ) / volume of unit cell
    RDF /= (rho*4*np.pi*r**2*dr)

    # Pad RDF and r with zeros for a nice looking plot
    r = np.insert(arr=r, obj=0, values=np.linspace(0,r[0],10))
    RDF = np.insert(arr=RDF, obj=0, values=[0]*10)

    return r, RDF, rho


def coord_numb(r, RDF, rho):
    ##### Find the first minimum #####

    # Use a median filtered version of the signal to get good estimates on the extremum indices
    filt_RDF = medfilt(RDF, 9)
    # Find the first and the halfway idx, and search inbetween
    first_max = np.argmax(filt_RDF)
    halfway_idx = int(len(filt_RDF)/2)
    min_idx = first_max + np.argmin(filt_RDF[first_max:halfway_idx])

    # Integrate up to that point - the padding does nothing, since it is zero
    shell_size = 4 * np.pi * np.trapz(y= r[:min_idx]**2 * rho * RDF[:min_idx], x=r[:min_idx])
    print(f'Coordination number: {shell_size:.4f}')  # Dimension    since integral of dimless RDF (just a histogram)
    # Return first solvation shell
    return first_max, halfway_idx, min_idx, shell_size

print("#### Task 2 - Calculate partial RDF ####")
# Load the trajectory from task 1
traj = Trajectory('../task1/mdTask1.traj')
traj_given = Trajectory('../task1/Na-aimd/NaCluster24.traj')

# Calculate the distances between the relevant species
eq_idx = 1000 # Index for equlibration
eq_idx_g = 8000
distances = load_distances('distances', traj[eq_idx:])
distances_given = load_distances('distances_given', traj_given[eq_idx_g:])

# Generate RDF
```

```python
r, RDF, rho = generate_partial_RDF(distances, len(traj[eq_idx:]))
r_g, RDF_g, rho_g = generate_partial_RDF(distances_given, len(traj_given[eq_idx_g:]))

# Calculate the first solvation shell of Na for both trajectories
first_max, halfway_idx, min_idx, _ = coord_numb(r, RDF, rho)
max_idx = [first_max, halfway_idx]
_, _, _, _ = coord_numb(r_g, RDF_g, rho_g)

# Plot
fig, ax = plt.subplots(figsize=(10,6))
ax.plot(r, RDF, linewidth=2, linestyle='-', alpha=1, label=r'Generated trajectory, $2\rm\, ps$.')
# ax.plot(r, medfilt(RDF, 9), linewidth=2, linestyle='-', label=r'Median filtered RDF, $2 \rm\, ps$.')
ax.plot(r_g, RDF_g, linewidth=2, linestyle='--', alpha=1, label=r'Given trajectory, $7 \rm\, ps$.')
# ax.scatter(r[max_idx], RDF[max_idx], marker='o', s=48, c='k')
# ax.scatter(r[min_idx], RDF[min_idx], marker='s', s=48, c='k')
ax.axhline(1, linewidth=2, linestyle='--', c='k', label=r'$g_{\rm Na^+O}=1$' )
ax.axvline(r[min_idx], linewidth=3, linestyle=':', c='C2', label=r'First min($g_{\rm Na^+O}$), $2 \rm\, ps$.')
ax.legend(loc='best')
ax.set_xlabel(r'$r$, (  )')
ax.set_ylabel(r'$g_{\rm Na^+O}$')
ax.set_xlim(0,6)
ax.grid()
plt.tight_layout()
plt.savefig('task2.png')
plt.show()

print("#### Task 2 - " +  "Finished ####".rjust(26))
```

A.3 `task3.py`

Please run in the folder ”task3”.

```python
# Built-in packages
import os.path

# External packages
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.signal import medfilt
from ase.io.trajectory import Trajectory
from tqdm import tqdm

# Set plot params
plt.rc('font', size=18)          # controls default text sizes
plt.rc('axes', titlesize=18)     # fontsize of the axes title
plt.rc('axes', labelsize=18)    # fontsize of the x and y labels
plt.rc('xtick', labelsize=18)    # fontsize of the tick labels
plt.rc('ytick', labelsize=18)    # fontsize of the tick labels
plt.rc('legend', fontsize=18)    # legend fontsize

'''
Calculate the RDF for the given water molecule trajectory in the same way as was done for task 2.
```

```python

This script is thus a slightly modified version of task2.py.
'''

def load_distances(file, trajectory):
    distances = [] # Holds all distances from Na+ for all snapshots
    if not os.path.exists(f'{file}_{len(trajectory)}_steps.npy'):
        print('---- Creating distances vector ----')
        for snapshot in tqdm(trajectory):
            # Get indices for Na+ ion and oxygen atoms
            O_idx = [O.index for O in snapshot if O.symbol=='O']
            visited_O = []
            for i, Oi in enumerate(O_idx):
                # For each O atom, get distance to all other O atoms - except those pairs that have already been visited
                # Hence, the last O idx will already have been checked and can be skipped
                if i < len(O_idx)-1:
                    visited_O.append(Oi)
                    O_other_idx = [idx for idx in O_idx if idx not in visited_O]
                    # Calculate their distances
                    distances.extend(snapshot.get_distances(Oi, indices=O_other_idx, mic=True))  # Enable minimum image convention to check pbc
        distances = np.array(distances)
        print('---- Saving to ' + f'{file}_{len(trajectory)}_steps.npy ----'.rjust(20))
        np.save(f'{file}_{len(trajectory)}_steps.npy', distances)
    else:
        print('---- Load from ' + f'{file}_{len(trajectory)}_steps.npy ----'.rjust(30))
        distances = np.load(f'{file}_{len(trajectory)}_steps.npy')
    return distances


def generate_partial_RDF(distances, n_snapshots):
    n = 200  # Number of points
    # Get a plot of the RDF from its histogram - it is accurate for sufficiently small bins
    RDF, b = np.histogram(distances, bins=n)  # Return the bin edges and use the as r vector for normalization
    RDF = RDF.astype(float) / n_snapshots  # Get the average occupation in each box over all snapshots
    RDF /= 11.5 # On average, we got the distance between 11.5 O-atoms; thus compensate for the overcounting
    r = np.array([(b[i-1]+b[i])/2 for i in range(1, len(b))])  # Position is middle point of each bin
    dr = r[1]-r[0]  # The size of the spherical radial shell

    # Normalize the obtained distribution function RDF(r) with the number density times the volume of the spherical shell at radius (r)
    # This would have been the RDF had the particles been uncorrelated (on average an rho particles per unit shell).
    V = traj[0].get_volume()
    rho = 24/V  # Number density of the relevant species of the system - ( 24 O + 1 Na ) / volume of unit cell
    RDF /= (rho*4*np.pi*r**2*dr)

    # Pad RDF and r with zeros for a nice looking plot
    r = np.insert(arr=r, obj=0, values=np.linspace(0,r[0],10))
    RDF = np.insert(arr=RDF, obj=0, values=[0]*10)

    return r, RDF
```

```python


def solv_shell(r, RDF):
    ##### Find the first minimum #####

    # Use a median filtered version of the signal to get good estimates on the extremum indices
    filt_RDF = medfilt(RDF, 9)
    # Find the first and the halfway idx, and search inbetween
    first_max = np.argmax(filt_RDF)
    halfway_idx = int(len(filt_RDF)/2)
    min_idx = first_max + np.argmin(filt_RDF[first_max:halfway_idx])

    # Integrate up to that point - the padding does nothing, since it is zero
    shell_size = np.trapz(y=RDF[:min_idx], x=r[:min_idx])
    print(f'First solvation shell: {shell_size:.4f}   ')  # Dimension    since integral of dimless RDF (just a histogram)
    # Return first solvation shell
    return first_max, halfway_idx, min_idx, shell_size

print("#### Task 2 - Calculate partial RDF ####")
# Load the trajectory from task 1
traj = Trajectory('../task1/Na-aimd/Cluster24.traj')
# Calculate the distances between the relevant species
eq_idx = 10000 # Index for equlibration
distances = load_distances('distances', traj[eq_idx:])

# Generate RDF
r, RDF = generate_partial_RDF(distances, len(traj[eq_idx:]))
first_max, halfway_idx, min_idx, _ = solv_shell(r, RDF)

# Plot
fig, ax = plt.subplots(figsize=(8,6))
ax.plot(r, RDF, linewidth=2, linestyle='-', alpha=1)
ax.axhline(1, linestyle='--', c='k')
# ax.legend(loc='best')
ax.axvline(r[min_idx], linestyle=':', c='k', label=r'First min($g_{NO}$), $2 \rm\, ps$.')
ax.set_xlabel(r'$r$, (  )')
ax.set_ylabel(r'$g_{OO}$')
ax.set_xlim(0,6)
ax.grid()
plt.tight_layout()
plt.savefig('task3.png')
plt.show()

print("#### Task 3 - " +  "Finished ####".rjust(26))
```