

Hands-On Session: Total Energy and Forces with Quantum ESPRESSO

Richard C. Remsing¹

*Department of Chemistry and Chemical Biology, Rutgers University, Piscataway,
NJ 08854*

I. OVERVIEW AND LEARNING GOALS

In this hands-on tutorial, we will learn how to setup and run DFT calculations of the total energy of a periodic system, followed by carrying out the basics for running molecular dynamics simulations. To do so, we will use the Quantum ESPRESSO (QE) software package, specifically the `pw.x` program. Traditional examples focus on solid silicon or water (liquid and/or solid) as model systems. Here, we will do something slightly different and model solid and liquid N₂, and even a polymeric phase of nitrogen. All the procedures and techniques are similar for other systems.

This hands-on exercise is organized as follows:

1. Total energy calculations
 - (a) Determining an appropriate wavefunction cutoff (`ecutwfc`)
 - (b) Determining the density cutoff (`ecutrho`)
 - (c) Determining the k-point mesh
 - (d) Computing the equation of state of bulk α -N₂
 - (e) Computing the equation of state of nitrogen in the black phosphorus structure
2. Molecular dynamics simulations
 - (a) Determining an appropriate timestep
 - (b) Effects of self consistent field (SCF) convergence
 - (c) MD simulations of solid and liquid N₂
 - (d) Visualizing and analyzing MD simulation results

Note: To make the calculations feasible within the time frame of the hands-on session, we will mostly work with system sizes much smaller than you would for publication quality calculations. For example, in many cases we will work with a unit cell containing 8 atoms (or if you are confident, you could use a $2 \times 2 \times 2$ supercell), and we will restrict our calculations to only a few **k**-points (if any at all). For higher quality calculations, use more **k**-points or significantly larger simulation cells.

For PAW pseudopotentials, in addition to the wavefunction cutoff `ecutwfc`, a second cutoff `ecutrho` controls the plane-wave expansion of the electron density and augmentation charges. While a ratio of `ecutrho/ecutwfc` = 4 is often used as a starting point, higher ratios (6–8 $\times$) are frequently needed for accurate forces and stresses with PAW. If you are familiar with CP2K or also doing that tutorial, this is analogous to the `REL_CUTOFF` parameter in CP2K’s Gaussian and plane waves (GPW) method, which controls how Gaussians are distributed across multiple density grids.

Move to the `Ecutrho` directory.

Within this directory, we are going to carry out the same steps as above, except we will vary the value of `ecutrho` with a fixed value of `ecutwfc`, which you determined in the previous exercise. Vary `ecutrho` from about 80 to 800 Ry with `ecutwfc` fixed at your converged value. In `analyze.sh`, set `ecutwfc` to your converged value. After that, run `SetupCalcs.sh` and then modify and submit the `submitRuns.sh` script.

Once the calculations have finished, run `analyze.sh` to produce the file `ecutrho_data.ssv`. In this file, we want to pay attention to two things: (1) How does the total energy (column 2) converge with `ecutrho`? (2) What is the ratio `ecutrho/ecutwfc` (column 3) at which the energy converges?

What ratio `ecutrho/ecutwfc` converges the energy? A ratio of 4–6 is typically sufficient for PAW.

You should use your converged values of `ecutwfc` and `ecutrho` in the next exercise, where you will determine the equation of state.

C. Determining the k-Point Mesh

For periodic systems, the Brillouin zone (BZ) must be sampled at a discrete set of **k**-points. The accuracy of the BZ integration improves with increasing mesh density. We will test convergence using Monkhorst-Pack meshes from $1 \times 1 \times 1$ (the Γ point) up to $4 \times 4 \times 4$.

Move to the `Kpoints` directory.

In `SetupCalcs.sh`, set `kpoint_values="1 2 3 4"` (these specify an $N \times N \times N$ Monkhorst-Pack mesh). The fixed `ecutwfc=80` Ry and `ecutrho=480` Ry are already set in the template. Run `SetupCalcs.sh`, `submitRuns.sh`, and `analyze.sh` to produce `kpoints_data.ssv`.

For insulating α -N₂, the Γ point is typically sufficient due to the large unit cell.

What mesh is needed to converge the energy to within 1 mRy?

D. Computing the Equation of State

In this exercise, we want to compute the equation of state (EOS) of solid α -N₂. We will assume that the EOS is compatible with the Murnaghan EOS,

$$E(V) = E_0 + K_0 V_0 \left[\frac{1}{K'_0(K'_0 - 1)} \left(\frac{V}{V_0} \right)^{1-K'_0} + \frac{1}{K'_0} \frac{V}{V_0} - \frac{1}{K'_0 - 1} \right], \quad (2)$$

where $E(V)$ is the total energy as a function of the cell volume, E_0 is the minimum energy, V_0 is the volume at the energy minimum, K_0 is the bulk modulus, and K'_0 is the derivative of the bulk modulus with respect to pressure. To compute the EOS of state, we will uniformly scale the lattice vectors of the system and compute the total energy as a function of the scaling parameter. We will then fit the resulting energy as a function of cell volume to the Murnaghan EOS to determine the various parameters.

Operationally, this exercise will be similar to the previous two. First, move to the `EOS` directory, where you will find files similar to those in the previous parts, except now there is an additional `GetEnergy.sh` bash script and a `PrintEnergyFile.pl` perl script, which replace the `analyze.sh` script. Within `SetupCalcs.sh`, `submitRuns.sh`, and `GetEnergy.sh`, we will need to specify the scaling parameters. The scaling parameter, let's call it λ , will multiply the lattice vectors, such that $\mathbf{a}_1 = (a, 0, 0)$ becomes $\lambda \mathbf{a}_1 = (\lambda a, 0, 0)$. In `SetupCalcs.sh`, specify 10–20 scaling parameters λ between 0.6 and 1.5 using the `scales=' '` variable. The script computes $L = \lambda \times 5.856$ Å and substitutes it into the `CELL_PARAMETERS` block of the template file `N2-template.in`. Once you've decided on the values of your scaling parameter λ and modified the appropriate scripts, setup and run your calculations.

When the calculations have finished, run the `GetEnergy.sh` script,

`bash GetEnergy.sh`

to produce the output file `Energy.dat`, which contains the scaling parameters in column 1 and the corresponding energies in column 2. You can fit this to the Murnaghan EOS using your favorite software. If that happens to be gnuplot, feel free to do so by running the gnuplot script `Fit.gnu` (`gnuplot Fit.gnu`). You can also enter the values

of your parameters into the `MurnaghanFit.txt` text file so that you don't forget them.

For this next part, move to the BP-EOS directory to compute the EOS for nitrogen in the black phosphorus structure. You're on your own for this one. The only hint I'll give is that you may need to increase the range of your scaling parameter so that it has an upper bound of 2 or so.

**How does your equation of state for BP-nitrogen compare to that of α -N₂?
Which phase is energetically more favorable?
Which has a larger volume per atom?**

III. MOLECULAR DYNAMICS SIMULATIONS

In this section, we will go through some of the basics of molecular dynamics (MD) simulations. To make the calculations feasible, we will use `ecutwfc` = 80 Ry and `ecutrho` = 480 Ry. When performing MD simulations, we need to choose an appropriate timestep for integrating the equations of motion, as well as an appropriate criterion for SCF convergence, to maximize efficiency and minimize error. Two systems are provided, one with a single 8-atom unit cell (UnitCell directory) and one with a $2 \times 2 \times 2$ supercell (SuperCell directory) composed of 64 atoms.

The exercises are generally set up similar to the previous ones — we will examine input files and prepare multiple sets of calculations using bash scripts.

A. BOMD

Born-Oppenheimer molecular dynamics (BOMD) simulations utilize a separation of electronic and nuclear degrees of freedom to simulate the time evolution of an atomic or molecular system. In this approach, the electronic structure problem is reduced to a *static* calculation, while the dynamics is performed on the nuclei, which are typically treated with classical mechanics. Thus, the electronic structure evolves in time only by its parametric dependence on the nuclei. The BOMD method is defined by the following equation of motion for the nuclear degrees of freedom:

$$m_I \ddot{\mathbf{R}}_I(t) = -\nabla_{\mathbf{R}_I} \min_{\Psi_0} \{ \langle \Psi_0 | \mathcal{H}_e | \Psi_0 \rangle \}, \quad (3)$$

where m_I is the mass of nuclei I with position $\mathbf{R}_I(t)$ at time t . Ψ_0 is the ground state electronic wave function and $\mathcal{H}_e$ is the electronic Hamiltonian, which satisfy the time-independent Schrödinger Equation

$$\mathcal{H}_e \Psi_0 = E_0 \Psi_0, \quad (4)$$

to be solved at each set of nuclear positions (time). Note that the right hand side of Equation 3 is the *force* on particle (nucleus) I , $\mathbf{F}_I$, due to the Born-Oppenheimer potential energy surface.

Other methods exist for performing *ab initio* MD simulations, which will not be addressed here. Ehrenfest MD, for example, attempts to solve the time-dependent electronic Schrödinger equation as the nuclei are evolved in time. Car-Parrinello MD (CPMD) is much more widely utilized, and is similar to BOMD but can have a lower computational cost in many situations. This is achieved by evolving both the nuclear and electronic degrees of freedom in time in a manner that keeps the electronic structure close to that in the ground state. For further details on these methods, see the book by Marx and Hutter, for example.

B. Choosing an appropriate timestep

1. Background

In order to perform a simulation of the dynamics in a system of particles, we must solve the equations of motion for the system of particles. For each particle J , Newton's equation of motion reads

$$m_J \ddot{\mathbf{R}}_J(t) = \mathbf{F}_J(\bar{\mathbf{R}}(t)) = -\nabla_{\mathbf{R}_J} V(\bar{\mathbf{R}}(t)), \quad (5)$$

where m_J and $\mathbf{R}_J$ are the mass and position of particle J , respectively, and t is the time. $\mathbf{F}_J$ is the force on particle J , which depends on the positions of all other particles in the system, $\bar{\mathbf{R}}$, and $V(\bar{\mathbf{R}})$ is the interaction potential, which also depends on $\bar{\mathbf{R}}$.

We will use the velocity Verlet algorithm to estimate the position of the particle at a future time $t + \Delta t$, where Δt is the *timestep*. This looks like a Taylor expansion in the positions accompanied by a velocity update in a second step,

$$\mathbf{R}(t + \Delta t) = \mathbf{R}(t) + \mathbf{v}(t)\Delta t + \frac{1}{2m}\mathbf{F}(t)\Delta t^2 \quad (6)$$

$$\mathbf{v}(t + \Delta t) = \mathbf{v}(t) + \frac{\mathbf{F}(t) + \mathbf{F}(t + \Delta t)}{2m}\Delta t, \quad (7)$$

where $\mathbf{v}(t)$ is the velocity of the particle at time t . It can be shown that the *global error* in the positions and velocities provided by these integration methods are $\mathcal{O}(\Delta t^2)$. Thus, the accuracy of our results critically depends on our choice of timestep Δt . Physically, the size of Δt is determined the highest frequency motion in the system, which we need to resolve. For water, for example, O-H bond vibrations dictate Δt . In this example, we will explore the dependence of the output of simulating N_2 on Δt .

Simulations will be performed in the *microcanonical* ensemble, that is, constant NVE , where N is the number of particles, V is the volume, and E is the energy. We will perform simulations for various values of Δt and monitor how well the energy E is conserved. For small enough Δt , E will fluctuate around a constant value, while E will fluctuate more and drift if Δt is too large.

2. Preparing the simulations

The necessary input files are in the `Template` directory. To run MD simulations in QE with `pw.x`, we set `calculation = 'md'` in the `&CONTROL` namelist of the input file `N2.in`. The timestep in QE is specified by the keyword `dt` in the `&CONTROL` namelist and is given in Rydberg atomic units. The conversion from femtoseconds is:

$$\Delta t_{\text{a.u.}} = \frac{\Delta t_{\text{fs}}}{0.04838}, \quad (8)$$

so that $1 \text{ fs} \approx 20.67 \text{ a.u.}$ The ensemble is controlled by the `ion_dynamics` keyword in the `&IONS` namelist: `ion_dynamics = 'verlet'` corresponds to the microcanonical (NVE) ensemble, while `ion_dynamics = 'langevin'` or the use of a Nosé-Hoover thermostat gives NVT behavior. The number of steps is set by `nstep` in `&CONTROL`.

Typically, when you setup a MD simulation, you need to specify an initial condition. Initial velocities can be chosen uniformly from a Gaussian distribution consistent with a desired temperature. The initial positions of the atoms can be more difficult to choose. Here, we start with the crystal structure of $\alpha\text{-N}_2$ determined experimentally. All MD simulations require an “equilibration” period, in which the system loses its memory of the initial conditions and observables like the energy, kinetic energy, and potential energy relax until they fluctuate around an average value. Some of the initial conditions you will use have already been equilibrated. Others have not. When they have already been equilibrated, you will start your simulation from a restart file instead of explicitly defining the coordinates in the input file.

3. Running the simulations

To run the simulations, we need to determine an accurate timestep by monitoring energy conservation as a function of the timestep. You should choose 3-5 timesteps. A reasonable range is between 0.1–5 fs. A good rule of thumb is that the timestep needs to be able to resolve the fastest or highest frequency motion in the system, typically a vibrational mode when the system contains bonds. Higher frequency modes require smaller timesteps to resolve their motion. So, for example, simulating H_2 or C-H bonds requires a smaller timestep than a simulation of N_2 .

Choose your 3-5 values of the timestep (in fs) and put those values into the `SetupCalcs.sh` bash script using the `timesteps=' '` variable. The script will automatically convert the values from fs to Rydberg atomic units before writing the QE input files.

Run the bash script with `bash SetupCalcs.sh`

This will setup a separate directory for each timestep that starts with `dt_` followed by the value of the timestep in fs (e.g., `dt_0.1`, `dt_0.5`, etc.).

To run the simulations, modify `submitRuns.sh` and submit this script to the queue.

At this point, *your jobs should be running*. The trajectory is written to the `N2-MD-pos-1.xyz` file, which can be visualized using VMD (visual molecular dynamics), for example.

QE does not write a separate energy file in the same format as CP2K; instead, thermodynamic information is printed directly to the output file. In particular, QE prints lines containing `Ekin + Etot (const)` for each MD step, from which the conserved quantity (total energy) can be extracted. Analyze the time-dependence of the total energy by running the python script `PlotEnergy.py`, which parses these lines from the QE output directly and will output two pdf files, `Energy.pdf` and `Energy-steps.pdf`, the latter of which contains the total energy as a function of timestep, not the physical time.

What can you say about energy conservation and its dependence on choice of timestep here? What value of Δt do you think is reasonable?

C. Energy Convergence

A crucial aspect of performing Born-Oppenheimer MD simulations is the accuracy with which the self-consistent electronic structure calculation step is carried out. In particular, the forces on each particle govern its motion and therefore the accuracy with which we sample the stationary distribution of the thermodynamic ensemble of interest. Thus, if the forces are not accurately converged, we will incur an error that affects the simulation in a manner similar to choosing too large a timestep. We demonstrate this issue here by performing a few short simulations and varying the tolerance for convergence of the SCF cycle.

1. Running the simulations

This exercise follows closely the previous one regarding timesteps. However, instead of varying Δt , we are changing the tolerance for the convergence of the SCF cycle, given in QE by the keyword `conv_thr` in the `&ELECTRONS` namelist. The parameter `conv_thr` is the threshold on the total energy change (in Ry) between successive SCF iterations; a typical range is 10^{-3} to 10^{-9} Ry. Choose a set of 3–5 values in this range and enter these values into the `SetupCalcs.sh` and `submitRuns.sh` scripts using the `conv_thr_values=' '` variable. Run the `SetupCalcs.sh` script and submit `submitRuns.sh` to run the jobs. Directories will be named `conv_1E-3`, `conv_1E-5`, etc. corresponding to each value. Note, the timestep is set to 1 fs, but you can change it to match what you decided on in the previous example (the plotting scripts may need to be changed).

After your simulations have completed, analyze the results in the same way that you did in the timestep example. You can plot the total energy as a function of time and tolerance using the python script `PlotEnergy-tol.py`, which will output the pdf file `Energy-tol.pdf` that compares the energy $E(t)$ as a function of time for each tolerance.

What can you say about energy conservation and its dependence on the accuracy of the electronic structure step?

Taking into account the results of the previous section, what combination of Δt and `conv_thr` do you think is reasonable?

D. Computing averages

1. Background

MD simulations are used to compute statistical mechanical averages of observables (not just to look at time series like we've done so far!). The average we wish to compute is the ensemble average $\langle X(\mathbf{R}) \rangle$ of some observable X which

depends on the positions (and more generally the momenta) of the particles in the system, $\bar{\mathbf{R}}$. To do so, we must adequately sample the phase space available to the system during our simulations. Therefore, to compute averages, we work under the *assumption of ergodicity*, which assumes that time averages are equivalent to ensemble averages:

$$\langle X(\bar{\mathbf{R}}) \rangle = \lim_{\tau \rightarrow \infty} \frac{1}{\tau} \sum_{t=1}^{\tau} X(\bar{\mathbf{R}}(t)). \quad (9)$$

In practice, we generate the time series $X(\bar{\mathbf{R}}(t))$ from our simulations, and then compute $\langle X(\bar{\mathbf{R}}) \rangle$ using Equation 9. The general issue of the accuracy of the *ergodic hypothesis* is important, but we will not discuss it further here.

2. Running the simulations

To compute averages, we will run two longer simulations, one for solid α -N₂ (**Solid**), one for liquid N₂ (**Liquid**), and a higher temperature liquid system (**HotLiquid**) to better illustrate the differences between solids and liquids. You will start with structures that have already been equilibrated in the canonical (NVT) ensemble at a constant temperature of 20 K for α -N₂ and 77 K for liquid N₂ so that simulations at constant energy should stay around those temperatures. Note: the liquid and hot-liquid simulations start from the crystal structure with randomized velocities at 77 K and 300 K respectively; a short equilibration period (first ~ 500 steps) should be discarded before computing averages. Your simulations will be in the microcanonical (NVE) ensemble. For the computation of dynamical properties, the NVE ensemble is typically preferred because there is no modification of the dynamics due to the presence of thermostats or barostats. In this exercise, we will compute thermodynamic, structural, and dynamic observables. You should be able to compare the results for the solid and liquid phase and distinguish between the two.

Move to the **Averages** directory. Setup and run simulation of the solid and liquid phase, within the corresponding directories **Solid**, **Liquid**, and **HotLiquid**. Make sure that the ***.in** file contains reasonable values for the timestep and SCF convergence tolerance, and make sure the simulations are at least 3 ps in length. We will use these simulations to gather statistics and compute averages. *Run the simulations by submitting the corresponding **submit.sh** script, appropriately modified.*

After your simulations have completed, run

```
bash prepare_analysis.sh
```

from within each simulation directory. This script parses the QE output and creates two files needed by the analysis codes: **N2-MD-pos-1.xyz** (the atomic trajectory in XYZ format) and **N2-MD-1.ener** (thermodynamic data including time, kinetic energy, potential energy, and total energy). Once these files have been created, you can analyze the results with the code provided in the **AnalysisCode** directory in the following sections.

3. Computing the average temperature

Position independent quantities provide a good introduction to the process of computing ensemble averages. In the microcanonical ensemble, the temperature is not a conserved quantity and fluctuates. We can therefore compute $\langle T \rangle$ following Eq. 9 to estimate the temperature at which our simulation has been performed. You can do this using the python script **ComputeTemperatureFlucs.py**,

```
python ComputeTemperatureFlucs.py
```

The script will print $\langle T \rangle$, $\langle (\delta T)^2 \rangle = \langle T^2 \rangle - \langle T \rangle^2$, and $\frac{C_v}{Nk_B}$ (discussed below) to the screen. The script will also create a plot, **Temperature.pdf**, that contains $T(t)$ (red line), $\langle T \rangle$ (black line), and the square root of the variance $\langle (\delta T)^2 \rangle = \langle T^2 \rangle - \langle T \rangle^2$ (grey shaded region).

Is your temperature roughly constant (fluctuating about an average value)?
What is the average temperature in your system?

In addition to statistical mechanical averages being connected to macroscopic quantities, like the temperature, fluctuations are also important and can tell us useful information about the system and its thermodynamics. For example, by examining fluctuations in the kinetic energy, and using the relationship between the kinetic energy and

the instantaneous temperature, we obtain a fluctuation-based expression for the constant volume heat capacity C_v ,

$$\frac{C_v}{Nk_B} = \frac{3}{2} \left[1 - \frac{3N}{2} \frac{\langle (\delta T)^2 \rangle}{\langle T \rangle^2} \right]^{-1}, \quad (10)$$

where N is the number of particles and k_B is Boltzmann's constant. Typically, such fluctuation relations require long trajectories and good statistics to converge. However, **you can make a preliminary estimate of C_v from your simulation using the $\langle T \rangle$ and $\langle (\delta T)^2 \rangle$ output by `ComputeTemperatureFlucs.py`.**

4. Pair correlation functions

Most characterizations of structure in condensed phases rely on computing position-dependent observables. A key quantity is the pair distribution function (or radial distribution function, RDF), $g(r)$, defined as

$$g(r) = \frac{1}{\rho_B^2} \left\langle \sum_i \sum_{j \neq i} \delta(\mathbf{r}_i) \delta(\mathbf{r}_j - \mathbf{r}) \right\rangle = \frac{V}{N^2} \left\langle \sum_i \sum_{j \neq i} \delta(\mathbf{r} - \mathbf{r}_{ij}) \right\rangle \quad (11)$$

where ρ_B is the bulk density and $\mathbf{r}_{ij}$ is the distance between particles i and j . Note that $g(r)$ is typically averaged over all particles of the same type, increasing statistics significantly.

Here, you can compute $g(r)$ in two ways. (1) You can load the trajectory (xyz file) into VMD and use VMD's radial distribution function calculator to compute $g(r)$ for you. To include periodicity in the calculation, you will need to enter the box dimensions into VMD's Tk Console and use this box size for all timesteps:

```

pbc set {5.856 5.856 5.856} -all
pbc box
pbc wrap -all

```

These three commands should create a cubic box of length 5.856 Å (a single unit cell), apply that box to all configurations, draw the (pbc box), and then “wrap” the atoms into the box for all frames. Once you set the pbc box dimensions, you can compute the RDF properly with VMD.

(2) Alternatively, you can use the fortran code provided, `CalcRDF.f`. To run the code, use the `RunRDF.sh` script within the directory of your simulation,

```
bash RunRDF.sh
```

which generates the output file `RDF.dat` containing the interatomic distance in column 1 and $g(r)$ in column 2.

Plot $g(r)$ for the liquid and the solid and compare them.

What differences do you observe between the liquid and solid $g(r)$?

What is the origin of these differences?

Note that the `CalcRDF.f` code only calculates $g(r)$ up to one half of the cubic box length, which is very small for a single unit cell. You might not be able to distinguish many differences between the liquid and solid RDF on this scale. As an illustrative example, your folders also contain RDFs for a $2 \times 2 \times 2$ supercell. Compare those as well and answer the same questions.

5. Dynamics: Time correlation functions, Green-Kubo relations, diffusion, and phonon density of states

A significant advantage of MD simulations is that we can compute dynamical quantities. The typical method of quantifying dynamics is through the computation of time correlation functions, which describe the decay of correlations in the system. Through formalisms like the Green-Kubo relations, one can relate time correlation functions to transport coefficients, such as the diffusion coefficient, conductivity, viscosity, and so on. In this exercise, we will examine the dynamics of liquid N_2 (and solid N_2) from your trajectory. In particular, the diffusion coefficient D is related to time correlations of the particle velocities,

$$D = \frac{1}{3} \int_0^\infty dt \langle \mathbf{v}(t) \cdot \mathbf{v}(0) \rangle, \quad (12)$$

where $\mathbf{v}(t)$ is the velocity of a particle at time t and the average includes an average over all particles in the system.

To compute the velocity autocorrelation function (VACF),

$$C_v(t) = \langle \mathbf{v}(t) \cdot \mathbf{v}(0) \rangle, \quad (13)$$

we need to know the velocity of each particle at each time in the trajectory. However, the `N2-MD-pos-1.xyz` file only contains the positions of the nuclei as a function of time. The code provided (`VACF_Positions.py`) will first compute the particle velocities from finite differences (though we could also just print the velocities during the simulation). You can run the VACF code by entering

```
python ../AnalysisCode/VACF_Positions.py ./ N2-MD-pos-1.xyz 1.0 N2_vacf.dat
```

The output file, `N2_vacf.dat`, contains the time in column 1 and the VACF in column 2. Plot and compare the VACF for liquid and solid N_2 .

Discuss the features in the VACF. What causes the decay?

If there is a minimum at short times, explain why.

What are the major differences between the liquid and the solid?

The phonon or vibrational density of states (VDOS) can be obtained from the Fourier transform of the velocity autocorrelation function $C_v(t)$,

$$\hat{C}_v(\omega) = \int dt C_v(t) e^{-i\omega t}, \quad (14)$$

where ω is frequency and $\hat{C}_v(\omega)$ is the VDOS. From the above expression of the diffusion coefficient as an integral of the VACF, we can show that D is related to the VDOS by

$$D = \frac{1}{3} \hat{C}_v(0). \quad (15)$$

The VDOS generally tells us about the frequencies of various dynamics in the system, e.g. vibrational frequencies. Compute the VDOS by entering

```
python VDOS_Positions.py ./ N2-MD-pos-1.xyz 1.0 N2_vdos.dat
```

The output file, `N2_vdos.dat`, contains the frequency in units of cm^{-1} in column 1 and the VDOS in column 2. Plot and compare the VDOS for liquid and solid N_2 . You probably want to consider a frequency range between 0 and 3000 cm^{-1} . Unfortunately, we might not have enough statistics to compute the VDOS completely accurately, due to our short simulation trajectories and small system size, but you can still try to make some conclusions (or run longer).

Approximately where is the highest frequency peak in the VDOS located and what physical motion does this correspond to?

What types of motions are contained in the lower frequency parts of the VDOS?

For each Green-Kubo relation, there exists a corresponding Einstein relation, which is valid at long times. In the case of self-diffusion, the Einstein relation is

$$\lim_{t \rightarrow \infty} \langle |\mathbf{r}(t) - \mathbf{r}(0)|^2 \rangle = 6Dt, \quad (16)$$

where the coefficient (here equal to 6) depends on the spatial dimensionality of the system. The quantity $\langle |\mathbf{r}(t) - \mathbf{r}(0)|^2 \rangle$ is referred to as the mean squared displacement, or MSD:

$$\text{MSD}(t) = \langle |\mathbf{r}(t) - \mathbf{r}(0)|^2 \rangle. \quad (17)$$

To obtain the diffusion coefficient, one typically performs a linear fit to the long-time, linear behavior of the MSD. Note that where the long-time behavior begins is better determined by observing $\text{MSD}(t)$ on a log-scale. At short times, $\text{MSD}(t) \sim t^2$ before crossing over to the long-time limit of $\text{MSD}(t) \sim t$. Many systems, and especially in glasses and other dynamically heterogeneous materials, the MSD can also display an intermediate *plateau* region associated

with caging of particles that prohibits them from diffusing on these timescales.

The MSD is computed from your simulation using `msdbl.k.f`. You can compile and run the code using the `RunMSD.sh` script

```
bash RunMSD.sh
```

which will output `msd_output`, containing the time in column 1 and the MSD in column 2. Compare the MSD for the solid and liquid phase.

What are the qualitative differences between the MSDs for liquid and solid N_2 ?

IV. ADVANCED SECTION: MOLECULAR DYNAMICS SIMULATIONS OF ION DIFFUSION IN SILVER IODIDE

If you’ve made it this far (or skipped ahead), you probably have some experience with MD simulations. As a slightly more advanced exercise, we are going to run BOMD simulations of α -AgI, a superionic conductor, using QE `pw.x`. We are also going to perform MD simulations of α -AgI using a neural network potential (NequIP) trained on the BOMD simulation data to predict atomic energies and forces at DFT accuracy. You can read more about these systems and DFT, neural networks, and empirical models of AgI in recent work, (Dhatarwal, Somni, and Remsing, *Nature Commun.* 2024, <https://www.nature.com/articles/s41467-023-44274-z>, and Dhatarwal and Remsing, arXiv 2025, <https://arxiv.org/abs/2504.16704>). You can read more about NequIP in the original paper (<https://www.nature.com/articles/s41467-022-29939-5>).

The goal of this exercise is to perform both sets of simulations and compare the results. You should use the same physical time (computer time) for each simulation, so that you can compare the relative efficiencies of the two approaches. To make things more interesting, the NequIP system uses a $3 \times 3 \times 3$ supercell, while the BOMD simulation uses a $2 \times 2 \times 2$ supercell (32 atoms, 16 Ag + 16 I) and/or a single unit cell (or both, your choice).

Go to the `AgI_Tutorials` directory to access the relevant files. The BOMD simulation files are in `1-AIMD`. The input file for the AIMD simulation is `agi-aimd.in`. To run the simulations, modify the input file to set (1) an appropriate timestep (`dt` in Ry a.u., $\approx 20.67 \times \Delta t_{fs}$), (2) SCF convergence (`conv_thr`), and (3) the supercell size. Unlike CP2K’s `MULTIPLE_UNIT_CELL` keyword, QE requires explicit atomic coordinates for the supercell. The provided `agi-aimd.in` uses a $2 \times 2 \times 2$ supercell (32 atoms, 16 Ag + 16 I) with all coordinates listed explicitly. If you want to explore a different supercell, make a separate directory and update the atomic positions accordingly. Once you have the input files setup, submit the job to the queue using `submit.sh`.

When the simulations are complete, run `python qe-2-extxyz.py AgI-MD.out AgI.extxyz` to convert the QE output to extended XYZ format for NequIP training (this replaces `cp2k-2-extxyz.py`). Then analyze the structure and dynamics of the system with VMD to compute the RDF and with the given python scripts to compute mean squared displacements, velocity autocorrelation functions, and vibrational densities of states. Also make note of how long the trajectory is for each simulation you’ve run.

Now we want to move to the neural network model. We don’t have time to train the model, but the code and training data used to generate the model is in the `2-training` directory if you’d like to try this on your own later. The necessary input files for the neural network model simulations are in `3-MLMD`, which stands for machine learning molecular dynamics.

To do the machine learning model-based MD simulations, we will use the software package LAMMPS, which is a package for classical mechanics-based molecular dynamics simulations. The input file for our ML-MD simulations is `in.nvt`. Much of the same qualitative inputs are similar to those for MD, except in a different format and without the sections about electronic structure. We need to specify an input configuration, here the `data.lmp` file, which also contains the box dimensions and the atom masses. Details regarding the DFT calculation needed for energy and forces is replaced by the definition of the model interactions, here the `pair_coeff` line. We also need to specify a timestep with the `timestep` command and details about how often to output thermodynamic properties and configurations in the trajectory. Finally, we are going to run the simulation at constant temperature, which is specified by the `fix 1 all nvt ...` command, and `run` specifies the number of timesteps to run in the MD simulation.

The parameters for the ML model determined by training the NequIP model are contained in `agi-full-model.pth`. This file is specified as input for the interactions between atoms with the `pair_coeff` line, which, for the nequip pair style, has the general format

```
pair_coeff * * model.pth <type name 1> <type name 2> ...
```

for different atom types.

You can run the ML-MD simulation using the `submit.sh` script. You will want to use a GPU to run simulations using ML models, because they more efficiently handle the evaluation of the ML model (if you try to train the model, you should also use a GPU).

When the simulations are complete, analyze the structure and dynamics of the system with VMD to compute the RDF and with the given python scripts to compute mean squared displacements, velocity autocorrelation functions, and vibrational densities of states. Also make note of how long the trajectory is for each simulation you've run.

How do the results of the ML model compare to the DFT model?

What is the difference in the amount of time you were able to simulate? Keep in mind that the ML system is significantly larger than the model used in the DFT simulations.