

Numerical Simulation and Scientific Computing II

Exercise 2

Submission is due by the end of Thursday 2022-05-05, 10 a.m. (submit through TUWEL).

- Include the name of all group members in your submission.
- Submit everything (your Python scripts, your own module files and any required input, resource or output files) as a single **zip** file per task.
- The Python scripts must work in a virtual environment created according to the instructions below, and with the command-line parameters described.
- The first line in any runnable script must be
`#!/usr/bin/env python`
- Try to format your code at least approximately following the [PEP 8 style guide](#). Using the [yapf](#) formatter is highly recommended – see below for some details. Always test your code after reformatting.

Setting up a virtual environment

When working with Python projects, it is often good practice to keep separate environments for each of them, to avoid interference and maximize flexibility, e.g. to keep different versions of a module for different projects. This section provides a set of quick instructions on how to create and use one of those virtual environments for this exercise.

1. Make sure that Python 3.8 or higher and pip are installed. For instance, in Ubuntu 20.04 with Python 3.8:
`sudo apt install python3-pip python3.8-venv`
2. Create a directory to keep everything tidy:
`mkdir ~/molecular_dynamics`
3. Move to the directory and create a virtual environment:
`cd ~/molecular_dynamics && python3 -m venv md`
4. Activate the environment:
`source md/bin/activate`
5. Install the required packages in the right order:
`pip install wheel`
`pip install numpy scipy matplotlib yapf`
`pip install jax jaxlib`
6. Optionally, if you want JAX to take advantage of your GPU and you have CUDA installed, you can run
`pip install --upgrade "jax[cuda]" \`
`-f https://storage.googleapis.com/jax-releases/jax_releases.html`

To leave the virtual environment, simply run
`deactivate`
To enter it again at any time, use the command

```
source ~/molecular_dynamics/md/bin/activate
```

When the virtual environment is active, the string `(md)` appears to the left of the prompt, and invoking `python` automatically calls the right interpreter.

The sequence of steps above also installs `yapf`, a popular Python code formatter, inside the virtual environment. It can be invoked as

```
yapf -i example1.py example2.py example3.py ...
```

to apply a nice format to an arbitrary number of Python files.

A toy molecular dynamics simulation

Your task in this exercise is to simulate the time evolution of M particles, each with a mass (m) of 18.998403 atomic mass units, by integrating Newton's equations of motion. Their potential energy is described using the Morse model:

$$E_{\text{pot}}(\mathbf{x}_1 \dots \mathbf{x}_M) = \sum_{i=1}^{M-1} \sum_{j=i+1}^M V_{\text{Morse}}(|\mathbf{x}_i - \mathbf{x}_j|), \text{ where} \quad (1a)$$

$$V_{\text{Morse}}(r) = D_e [e^{-2\alpha(r-r_e)} - 2e^{-\alpha(r-r_e)}]. \quad (1b)$$

\mathbf{x}_i is the position of atom i in 3D space, and the values of the parameters of the model are:

$$\begin{aligned} D_e &= 1.6 \text{ eV} \\ \alpha &= 3.028 \text{ \AA}^{-1} \\ r_e &= 1.411 \text{ \AA} \end{aligned}$$

This is an exceedingly poor approximation to a real system from the standpoint of realism, but it is very simple to implement and runs fast enough not to require any parallelization.

The force on atom i (denoted as \mathbf{f}_i) is just the gradient of E_{pot} with respect to \mathbf{x}_i with its sign changed, and the corresponding acceleration is computed as $\mathbf{a}_i = \mathbf{f}_i/m$. Use the velocity Verlet algorithm to integrate the equations of motion:

$$\mathbf{x}_i(t + \Delta t) = \mathbf{x}_i(t) + \mathbf{v}_i(t)\Delta t + \frac{1}{2}\mathbf{a}_i(t)(\Delta t)^2 \text{ and} \quad (2a)$$

$$\mathbf{v}_i(t + \Delta t) = \mathbf{v}_i(t) + \frac{\mathbf{a}_i(t) + \mathbf{a}_i(t + \Delta t)}{2}\Delta t, \quad (2b)$$

where \mathbf{v}_i is the velocity of atom i and Δt is the time step.

Use periodic boundary conditions corresponding to a cubic simulation box with side L , along with the minimum image convention.

File format

Your code must be able to read and write snapshots of the simulation box in the following ad-hoc, text-based format:

Line 1: Number of atoms (integer)

Line 2: Arbitrary comment/description (free format)

Line 3: Box side length (decimal number)

Line 4: $x_1 y_1 z_1 v_1^{(x)} v_1^{(y)} v_1^{(z)}$ (six decimal numbers)

Line 5: $x_2 y_2 z_2 v_2^{(x)} v_2^{(y)} v_2^{(z)}$ (six decimal numbers)

\vdots

Line M+3: $x_M y_M z_M v_M^{(x)} v_M^{(y)} v_M^{(z)}$ (six decimal numbers)

Any number of spaces and tabs can appear around and between fields, and are not considered significant. Integers and decimal numbers are defined as anything the Python `int()` and `float()` built-ins can parse, respectively.

Task 1: Questions (2 points)

- Describe a suitable and consistent system of units for molecular dynamics, and specifically the units of length, mass, velocity, energy and time.
- Explain the advantages of automatic differentiation as a method to calculate forces.
- When the potential energy of a system is described using the Morse model, how does the number of calculations per time step scale with the number of particles? Why?
- Describe a strategy to make that number of calculations proportional to the number of particles.

Task 2: Generation of initial conditions (3 points)

Create a Python script that takes three command-line arguments, namely

- M , a number of particles;
- L , a side length for the simulation box, in Å; and
- T , a temperature in K used to generate the initial conditions

and generates a "relaxed" (i.e., low-energy) starting configuration for a molecular dynamics simulation. Specifically, the script must:

- Start by placing the M particles at random points in the box. The distribution needs not be uniform – in fact, it is desirable to avoid placing two particles too close together.
- Move the particles to a local energy minimum using the conjugate-gradients (CG) minimizer implemented as part of the `scipy.optimize.minimize` function.
- Draw random velocities from a Gaussian distribution with zero mean and a standard deviation $\sqrt{k_B T/m}$, where k_B is the Boltzmann constant (use `numpy.random.multivariate_normal`).
- Obtain the average velocity $\bar{\mathbf{v}} = \frac{1}{M} \sum_{i=1}^M \mathbf{v}_i$ and subtract it from each individual velocity, i.e., assign

$$\mathbf{v}_i \leftarrow \mathbf{v}_i - \bar{\mathbf{v}},$$

so that the velocity of the center of mass of the system is zero.

- Save the result in the format described above.

To complete this task you must implement the calculation of the potential energy and the forces in the Morse model. For the latter, use automatic differentiation as implemented in the high-performance `jax` library. As a quick sanity test, check that the sum of all forces along each of the Cartesian axes is close to zero.

Task 3: Trajectory generation (2 points)

Write another Python script that takes three arguments:

- `input.xyz`, the path to a file with a snapshot of the system;
- Δt , a time step; and
- N , a number of time steps.

The script must integrate Newton's equations of motion for the system starting with the initial conditions provided, for a total of $N - 1$ time steps of length Δt , and store the resulting trajectory as a single file where each frame is written directly after the previous one in the format described in the introduction.

Task 4: Post-processing (3 points)

Write a third Python script that takes, as its single command-line argument, the path to a trajectory file such as the one generated by the previous script, and generates two outputs:

- A text file with as many lines as time steps are stored in the input trajectory and two columns containing the following data for each step:
 1. The potential energy E_{pot} .
 2. The kinetic energy $E_{\text{kin}} = \frac{m}{2} \sum_{i=1}^M |\mathbf{v}_i|^2$.
- A second text file with two columns:
 1. The first column contains the values of a variable r sampled at regular intervals from 0 to $L/2$.
 2. The second column contains the fraction of particle pairs at a distance of at most r , averaged over the trajectory but discarding the first 25% of it.

Using the three scripts, complete these steps:

1. Create a cubic simulation box with a side length of 15 Å, 1000 particles, and initial velocities corresponding to a temperature of 300 K.
2. Run a molecular dynamics simulation for one million steps. Check the conservation of the mechanical energy $E_{\text{pot}} + E_{\text{kin}}$; tweak the time step and repeat the run until it is conserved.
3. Create a plot of the evolution of the kinetic temperature $E_{\text{kin}}/(\frac{3}{2}Mk_{\text{B}})$ over time during the simulation and discuss what is happening.
4. Using only the second half of the trajectory, estimate the specific heat of the system using the formula

$$c_v = k_{\text{B}} \frac{\langle E_{\text{kin}} \rangle^2}{\sigma_{E_{\text{kin}}}^2}, \text{ where } \sigma_{E_{\text{kin}}}^2 = \langle (E_{\text{kin}} - \langle E_{\text{kin}} \rangle)^2 \rangle.$$