Institute of Materials Chemistry

nssc@iue.tuwien.ac.at

Numerical Simulation and Scientific Computing II
Exercise 3

Submission is due on Monday 2021-05-06, 10 a.m. (submit to nssc@iue.tuwien.ac.at)

- Include the name of all group members (max. 3) 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 3.7 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
   pip install jax jaxlib yapf
   ```

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 enviroment 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 enviroment. 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 of Ar, a noble gas, by integrating Newton's equations of motion. The potential energy of the gas is described using the Lennard-Jones model:

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

$$V_{\text{LJ}}(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right], \tag{1b}$$

$\boldsymbol{x}_i$ is the position of atom i in 3D space and σ and ε are parameters of the potential. This is an exceedingly poor approximation from the standpoint of realism, but it is very simple to implement and runs fast enough not to require any parallelization. To make the exercise even simpler, operate in a natural system of units where $\varepsilon = 1$, $\sigma = 2^{-1/6}$ and the atomic mass of Ar (m) is taken as 1. Therefore, the force on atom i (denoted as $\boldsymbol{f}_i$) is just the gradient of E_{pot} with respect to $\boldsymbol{x}_i$ with its sign changed. Use the velocity Verlet algorithm to integrate the equations of motion:

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

$$\boldsymbol{v}_i(t + \Delta t) = \boldsymbol{v}_i(t) + \frac{\boldsymbol{f}_i(t) + \boldsymbol{f}_i(t + \Delta t)}{2}\Delta t, \tag{2b}$$

where $\boldsymbol{v}_i$ 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)

⋮

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.

Remarks about design

You have almost total flexibility when designing your solution. However, a good modular design will greatly facilitate debugging, improve readability and, perhaps more importantly, allow you to share code between the different parts of the exercise. Whenever possible, try to use vectorized operations instead of loops to obtain better performance.

Task 1: Questions (2 points)

(a) Is the fourth-order Runge-Kutta method a good integrator in the context of molecular dynamics? Why? Give at least two reasons.

(b) Describe the advantages of automatic differentiation as a method to calculate forces.

(c) When the potential energy of a system is described using the Lennard-Jones model, how does the number of calculations per time step scale with the number of particles? Why?

(d) 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; and
- Σ, a standard deviation for the velocity distribution

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 the requested standard deviation (use `numpy.random.multivariate_normal`).
- Obtain the average velocity $\bar{\boldsymbol{v}} = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{v}_i$ and subtract it from each individual velocity, i.e., assign

  $$\boldsymbol{v}_i \longleftarrow \boldsymbol{v}_i - \bar{\boldsymbol{v}},$$

  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 Lennard-Jones 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 any of the Cartesian axes is 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 $E_{\text{kin}} = \frac{1}{2} \sum_{i=1}^{M} |\boldsymbol{v}_i|^2$.
- 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 an estimate of the average volumetric density of particles at a distance r from the origin of coordinates when one particle is at the origin. This must be obtained by averaging over the trajectory but discarding the first 25% of it.

Using the three scripts, study the conservation of the total energy $E_{\text{pot}} + E_{\text{kin}}$ as a function of the time step, and then analyze the number of time steps needed to obtain a converged value of the time-averaged kinetic and potential energies. Create plots illustrating these observations. Use $M = 200$ particles, $\Sigma = 1$ and a value of the box side length L such that there are on average 0.8 particles per unit volume.

Finally, plot the contents of the second file generated by the program and discuss how the results would change if there were no interactions between the particles ($E_{\text{pot}} = 0$).