Exercise 3
Particle3D Class, Time Integration

Computer Modelling 2017-18

Due: 5pm Thursday, Week 11, Semester 1

1 Aims

In this exercise, you will put into practice concepts of object-oriented programming, by first writing a new *class* that describes point particles moving in 3D space, and then using that class to perform simple *time integration* of a pair of particles to simulate spinning and vibrating N_2 and O_2 molecules.

Your code will analyse the particles' trajectories to obtain vibrational frequencies that can be compared to experimental data. We also ask you to write a short group report analysing the performance of your time integration code, which serves as an exercise for the project reports due towards the end of this course.

2 Tasks

2.1 `Particle3D` Class

Write a Python class called `Particle3D` that describes a point-like particle moving in 3D space. It should have the following capabilities:

- Properties to hold the particle's mass (a number), label (a string), position (a `NumPy` array), and velocity (a `NumPy` array).
- A `__init__` method that initialises those properties.
- A `__str__` method that prints the particle in the following format:

  ```
  <label> <x-pos> <y-pos> <z-pos>
  ```

- A method to return an object's kinetic energy.

Table 1: Simulation parameters for O_2 and N_2.

	m [a.u.]	D_e [eV]	r_e [Å]	α [Å^{-1}]
O_2	16.00	5.21322	1.20752	2.65374
N_2	14.01	9.90523	1.09768	2.68867

- A method to update the velocity of a particle for a given timestep and force vector, as
$$\boldsymbol{v}(t+dt) = \boldsymbol{v}(t) + dt \cdot \boldsymbol{f}(t)/m \tag{1}$$
- A method for a first-oder update of the particle position for a given timestep, as
$$\boldsymbol{r}(t+dt) = \boldsymbol{r}(t) + dt \cdot \boldsymbol{v}(t) \tag{2}$$
- A method for a second-order update of the particle position for a given timestep and force, as
$$\boldsymbol{r}(t+dt) = \boldsymbol{r}(t) + dt \cdot \boldsymbol{v}(t) + dt^2 \cdot \boldsymbol{f}(t)/2m \tag{3}$$
- A static method to create a particle from a file entry. It should take a file handle as argument and return a `Particle3D` object.
- A static method that returns the relative vector separation of two particles.

2.2 Two-particle Simulation

Write *two* test programs that perform time integration for two particles' motion. One program should use the symplectic Euler time integration algorithm, the other should use the velocity Verlet time integration algorithm.

Both codes should plot, as function of time, the relative separation of the two particles and the total energy of the system.

Your simulation should consist of two particles interacting via a Morse potential. The potential settings (see Table 1) should be read from an input file. The following initial conditions should also be read from the input file:

- Particle 1: $\boldsymbol{r}_1 = (1/2r_e, 0, 0)$, $\boldsymbol{v}_1 = (0.1, 0, 0)$.
- Particle 2: $\boldsymbol{r}_2 = (-1/2r_e, 0, 0)$, $\boldsymbol{v}_2 = (-0.1, 0, 0)$.

2.3 Report

Write a *short* group report (about 2 pages) on the results and performance of your programs. List the names of your group members at the beginning of the document.

For the report, run simulations of the N_2 molecule with different timesteps dt. Estimate, for both time integrators, the maximum time step dt_{max} that can be used to simulate the system with a relative energy fluctuation of less than $\Delta E/E = 10^{-3}$. Which algorithm performs better for a given timestep dt and, conversely, which algorithm allows to use larger timesteps with the same relative energy error?

Using the velocity Verlet integrator, and with the time step set to dt_{max}, determine the vibrational frequencies for both N_2 and O_2 by analysing the period of oscillation of the particle separation. Compare the results to experimental values, $\nu(N_2) = 2359\ \mathrm{cm}^{-1}$ and $\nu(O_2) = 1580\ \mathrm{cm}^{-1}$, and to the values you would expect from the interaction potentials assuming small amplitudes.

Now put a spin on the molecules by setting the initial velocities to $\boldsymbol{v}_1 = (0.1, 0.3, 0.0)$ and $\boldsymbol{v}_2 = (-0.1, -0.3, 0.0)$. How do the frequencies $\nu(N_2)$ and $\nu(O_2)$ change and why?

In the report, include representative plots of the particle separation as function of time, for both molecules and both sets of initial conditions.

3 Detailed Instructions

You may find it easiest to base your programs on the 1D examples provided. The instructions below are a suggestion of how to gradually build your code based on those examples. You do not have to follow this suggested route, but beware that TA's and lecturers can best help you with issues and debugging if you do not stray too far from it.

3.1 Starting Point: 1D Time Integration

Make sure you are familiar with the symplectic Euler and velocity Verlet time integration algorithms described in the background material and the lecture. Download and read the example codes `Particle1D.py`, `symplecticEuler1D.py`, and `velocityVerlet1D.py` from LEARN. Run both the `symplecticEuler1D` and `velocityVerlet1D` programs. You should provide an output filename as an argument to both codes:

```
[user@cplab001 ~]$ python symplecticEuler1D.py euler.dat
[user@cplab001 ~]$ python velocityVerlet1D.py verlet.dat
```

Learn how to save records of the trajectory and energy plots produced by the code (using the `matplotlib` package), or use the output file together with another plotting program, such as `xmGrace`:

```
[user@cplab001 ~]$ xmgrace -nxy euler.dat &
```

Make sure you understand the codes and confirm that both algorithms can simulate the motion of a particle in a double well potential.

3.2 `Particle3D` Class

You can base your `Particle3D` class on the one-dimensional equivalent that is provided, `Particle1D`. The class you write now will be reused for the project work in this course, so it is important that it is correct and works well. Implement the functionality requested above, then make sure that your `Particle3D` class can be run by Python without any error messages before you proceed to the next stage:

```
[user@cplab001 ~]$ python Particle3D.py
```

3.3 The Morse potential

Your programs should include methods that compute the potential energy and pair-wise forces between particles interacting through the Morse potential. The Morse potential is a commonly used pair-potential to describe covalent bonds in molecules or clusters. Its expression for the energy of two particles at $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ is

$$U_M(\boldsymbol{r}_1, \boldsymbol{r}_2) = D_e \left((1 - \exp\left[-\alpha(r_{12} - r_e)\right])^2 - 1 \right) \tag{4}$$

where $\boldsymbol{r}_{12} = \boldsymbol{r}_1 - \boldsymbol{r}_2$ and $r_{12} = |\boldsymbol{r}_{12}|$. The force on the particle at $\boldsymbol{r}_1$ is

$$\boldsymbol{F}_1(\boldsymbol{r}_1, \boldsymbol{r}_2) = -2\alpha D_e (1 - \exp\left[-\alpha(r_{12} - r_e)\right]) \exp\left[-\alpha(r_{12} - r_e)\right] \boldsymbol{r}_{12} \tag{5}$$

and $\boldsymbol{F}_2 = -\boldsymbol{F}_1$.

The parameters D_e, r_e, and α are system-specific. They relate to the depth, position, and curvature of the potential minimum, respectively, see Figure 1.

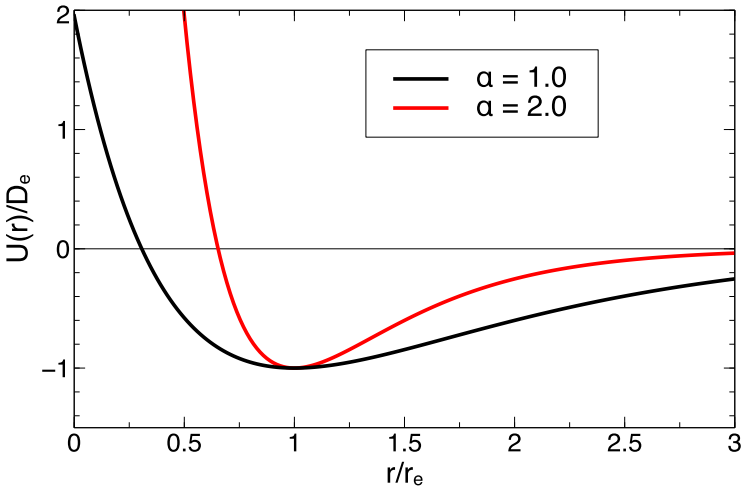

Figure 1: The Morse potential, shown in relative distance (r/r_e) and energy (U/D_e) co-ordinates, and for two different values of α.

3.4 Time Integration Codes

We recommend you base these codes on the example 1D versions provided. You will need to adapt the force and energy calculations to consider interactions with the Morse potential. Write methods to evaluate both quantities and use them during the time integration. Note that your force will depend on the particles' *instantaneous* positions and will need to be re-computed at every timestep. You will also need to handle two instead of one moving particles.

The total energy is the sum of kinetic and potential energies,

$$E_{tot} = T_1 + T_2 + U_{12} = \frac{1}{2}m_1|\boldsymbol{v}_1|^2 + \frac{1}{2}m_2|\boldsymbol{v}_2|^2 + U_M(\boldsymbol{r}_1, \boldsymbol{r}_2) \tag{6}$$

Beware of the vector character of most quantities – as opposed to scalar in 1D. Make use of the methods you implemented in `Particle3D` wherever possible.

3.5 Units

The unit system of the simulation is implicitly defined and completely fixed by choice of the input parameters. By using the quantities as given in Table 1, all energies are in electron volts ($1\text{eV} \approx 1.602 * 10^{-19}$ J), all lengths in Ångstroms ($1\text{Å} = 10^{-10}$ m), and all masses in atomic units ($1\text{u} \approx 1.66054 * 10^{-27}$ kg). This means that the unit of time, [t] = [L]*([m]/[E])$^{1/2}$, is very small, about 10.18 fs.

Vibrational spectroscopic data is usually given in units of wave numbers (or "inverse centimeter", or "cm^{-1}"), and measures inverse wave length, λ^{-1}. With $c = \lambda f$, a vibration of 1 cm^{-1} corresponds to a frequency of $f = 2.99792458 * 10^{10}$ Hz, or a period of $T \approx 33.36$ ps.

4 Submission

Package the subdirectory that contains `Particle3D.py`, your simulation programs, and your report. For instance, if your subdirectory is called `exercise-3`, you can use the `tar` and `gzip` commands by running:

```
[user@cplab001 ~]$ tar cvf ex3.tar exercise-3
[user@cplab001 ~]$ gzip ex3.tar
```

One of your group members should submit the compressed package (e.g., `ex3.tar.gz`) through the course LEARN page, by **5pm on Thursday, week 11 of semester 1**. Submissions from individuals will not be accepted.

5 Marking Scheme

This assignment counts for 15% of your total course mark.

1. `Particle3D` Class [10]

- Properties and initialisation are correct [2]
- Methods are correct [5]
- Code layout and comments are clear, logical [3]

2. Particle Simulations [10]
    - Symplectic Euler and velocity Verlet implemented correctly [4]
    - Force and energy laws implemented correctly [3]
    - Code layout and comments are clear, logical [3]
3. Report [10]
    - Timestep estimates [1]
    - Vibrational frequencies for two initial conditions [2]
    - Plots of particle separations [1]
    - Discussion of results and conclusions [6]

Total: 30 marks.