Assignment 6

Problem 1

In this assignment, you will build a Python module to model a carbon monoxide (CO) molecule, which will be used for performing classical mechanical simulations of the motion of the atoms of CO over time.

For simplicity, we assume the molecule is constrained to move along the x-axis. The interaction between the carbon and oxygen atoms is described by a Morse potential (see also Problem 1 of assignment 4):

$$V(r) = D_e \left(1 - e^{-\alpha(r - r_e)}\right)^2,$$

where r is the bond length, $r = x_O - x_C$ with $x_O > x_C$, $D_e = 11.22$ eV is the well depth, $r_e = 1.128$ Å is the equilibrium bond length, and $\alpha = 2.594$ Å^{-1} controls the width of the potential.

(a) Create a Python module `model_potential.py` containing a class `CO_morse` to model the CO molecule and a function `morse_potential` returning the energy and force for the Morse potential for given values of the bond length r. The class should include the following methods:

- An `__init__` constructor setting as attributes a (2,) NumPy array of x-positions `[x_C, x_O]`, a (2,) NumPy array of velocities `[v_C, v_O]`, and a (2,) NumPy array of atomic `masses [12.01, 16.00]` (atomic mass units).
- `calculate_energy_and_forces`, which calculates the bond length r, the potential energy $V(r)$, and the forces on the atoms, $F(x_C)$ and $F(x_O)$, from the `morse_potential` function.
- `calculate_kinetic_energy`, which computes and returns the total kinetic energy of the molecule, $E_{\text{kin}} = \frac{1}{2} m_C v_C^2 + \frac{1}{2} m_O v_O^2$.
- `get_total_energy`, which returns the sum of the potential and kinetic energy.

(b) In a Jupyter notebook import your `model_potential` module. Create an array of bond lengths from 0.7 Å to 2.5 Å, and for each bond length calculate the potential energy and the force on the carbon atom. Plot the potential energy $V(r)$ and the force $F_C(r)$ vs. r. Briefly comment on the relationship between the energy curve and the force plot.

(c) At the end of your module, include a test block that runs the following sanity checks: (i) At $r = r_e$ confirm that $F_{\text{C}} = 0$, and (ii) $F_{\text{C}} + F_{\text{O}} = 0$ for random $(x_{\text{C}}, x_{\text{O}})$ with $x_{\text{O}} > x_{\text{C}}$. Run `model_potential.py` as a script and verify that these relations are satisfied.

(d) A good way to verify that the analytical forces are implemented correctly is to compare them with forces computed numerically using finite differences. In a Jupyter notebook, calculate the analytical force on the carbon atom with a random bond length of $r = 1.3$ Å and zero velocities. Calculate the numerical force using the central difference formula:

$$F_{\text{num}} = -\frac{V(r+\delta) - V(r-\delta)}{2\delta}$$

where δ is a small number (e.g., 10^{-5} Å). Confirm that the analytical and numerical forces agree to within a small tolerance.

Problem 2

In this problem, you will implement a module for performing simulations of the vibrations of the C-O bond of the carbon monoxide molecule via classical mechanical equations of motion. The C-O bond is modelled with a Morse potential, and the equations of motion are integrated using the *velocity Verlet* algorithm. Assume the CO molecule is constrained to move along the x-axis and that $x_O > x_C$.

(a) Create a Python module `molecular_dynamics.py` that contains a class `MD` to propagate a CO trajectory. The class should take as inputs the initial atom positions `[x_C, x_O]`, velocities `[v_C, v_O]`, the atomic masses `[12.01, 16.00]`, the time step `dt`, and the `CO_morse` class of the `model_potential.py` module from Problem 1 as a calculator. It should then implement the following methods:

- `velocity_verlet`, which performs *one* integration step using velocity Verlet, calling the `CO_morse` class (the calculator) to get the forces at the needed positions:

$$x_i(t+h) = x_i(t) + h\,v_i(t) + \frac{h^2}{2m_i}\,F_i\big(x_i(t)\big),$$

$$v_i(t+h) = v_i(t) + \frac{h}{2m_i}\Big[F_i\big(x_i(t+h)\big) + F_i\big(x_i(t)\big)\Big],$$

  where h is the time step.

- `run`, which takes as input the number of steps `nsteps`, and runs a full trajectory propagation, stores and returns arrays of $x_C(t)$, $x_O(t)$, $E_{\text{kin}}(t)$ and $E_{\text{pot}}(t)$ for subsequent analysis/plotting.

IMPORTANT

To keep units consistent, $a = F/m$ needs to be converted from eV/Å·amu to Å/fs^2. To do that, multiply F/m by 0.009648 inside the velocity Verlet code.

To convert the kinetic energy from amu·Å^2/fs^2 to eV multiply by 103.650907.

(b) In a Jupyter notebook import the `molecular_dynamics.py` and `model_potential.py` modules. Set the initial velocity for C and O as zero ($v_C = v_O = 0$) and the initial positions as $x_C = 0$, $x_O = 1.5$ Å. Then, run two simulations, one with a short time step `dt`$= 0.01$ fs and one with a long time step `dt`$= 5$ fs. Choose `nsteps` such that both simulations run for 20 fs. For each run, plot $E_{\text{kin}}(t)$, $E_{\text{pot}}(t)$, and $E_{\text{tot}}(t)$ on the same figure. Briefly discuss how `dt` affects the outcome and stability of the simulation.

(c) Set $v_C = v_O = 0$, $x_C = 0$, and run simulations for the initial bond lengths $r = 0.85$, 1.00, 1.128, 2.00 Å for a total duration of the trajectory of 50 fs. For each case, plot $x_C(t)$, $x_O(t)$, $r(t)$ as a function of time in the same figure. Briefly describe what you observe for each simulation. Can you explain the different results? Does the molecule dissociate for any of the initial guesses?

(d) Start from $r_{\text{CO}} = r_e$ and assign different initial velocities to the atoms (e.g. try $v_C = \pm 0.05$ and $v_O = \mp 0.05$ Å/fs). Run simulations for up to 50 fs. For each case, plot $x_C(t)$, $x_O(t)$, $r(t)$ as a function of time in the same figure. Briefly explain the results of the simulations.