Spr4 - Classes
Ideal Gas Simulation with OOP

Poynting's Python Society

26th February 2022

Abstract

This worksheet will walk through creating a simple molecular dynamics simulation using Python classes to represent individual particles diffusing from a point in 2D.

1 Introduction

The task for this week is to create a simple 2D molecular dynamics simulation of an ideal gas diffusing away from a point. Each particle in the gas should be represented by an individual instance of a class and the simulation should be 'run forward' in time by looping through each instance and calling some kind of `time_step` method.

2 The Particle Class

Start by creating the class to represent an individual particle.

It should have two key attributes: `pos` and `vel` - these should be numpy arrays representing the position and velocity vectors of the particle respectively. NumPy arrays are best here as you can do speedy maths with them :).

These variables can be, for the moment, randomly initialised in the constructor (the `__init__` bit), or you might want to specify the initial conditions yourself using positional/keyword arguments.

The class should have a method that is able to work out the particle's position at the next time step and reassign the position attribute. That is given a small time Δt, the position at the next time step is given by:

$$\vec{r}(t + \Delta t) = \vec{r}(t) + \vec{v}\Delta t \tag{1}$$

3 Initialising the Gas

Initialise 10 particles by creating new instances of the particle class you've just written.

You might want to specify *where* the particles start (e.g. the origin or randomly, etc.) - how can you do this? How should you store each of the instances of the particle class? You could make 10 different variable names for each instance (e.g. `particle1, particle2` etc., is there a better way? (e.g. storing them as items in a list?).

Plot the positions of all the particles on a matplotlib figure to make a sense-check.

4 Running the Simulation

Run the simulation 'forward' in time by looping through every instance of the particle class and calling the appropriate method. After all particles have 'stepped forward', plot the results.

Once you've done this a few times, you might want to write a function that does all this stuff for you; repeatedly calling this function should 1) perform all time-stepping calculations 2) plot the positions of all particles after the time-step.

All done :)

5 Extension: Gas in a Container

As an extension: how could you account for the gas being contained within a finite volume? You'll need to write some new methods for the particle class which a) check whether the particle is 'inside' the contained after a given time-step calculation, b) make corrections on the position *and* velocity of any particles found to be outside the container after a given time-step. (i.e. reverse the velocity, make sure new position is inside the container.)