Tobias Neckel Max-Planck, May 2019

Python Tutorial: Object-oriented programming

1) Modeling simple particles

- Create a class `Particle` that models a simple 2D particle with the following attributes:

  1. A mass m
  2. An x coordinate
  3. A y coordinate

  These three attributes should be set in the constructor.

- Add a method `info()` that prints the particle's attributes to the command line (you are free to choose how they are displayed).

- Assume two particles p_1 and p_2 interact via a Lennard-Jones potential given by

$$V(r) = 4 \cdot \left[\left(\frac{1}{r} \right)^{12} - \left(\frac{1}{r} \right)^{6} \right],$$

  where r is the distance between the two particles. Write a function `potential(self,p2)` that calculates and returns the Lennard-Jones potential between the current particle (data accessible via `self`) and a second particle `p2`.

- Finally, write a function that allows to add two objects of the class `Particle`. The result of adding two particles should be another particle whose mass is the sum of the individual particle's masses, and whose position $\vec{r}_s$ is given by the center of mass of the two particles:

$$\vec{r}_s = \frac{\sum_i m_i \vec{r}_i}{\sum_i m_i}$$

2) Quadrature rules

We are interested in approximating integrals of one-dimensional functions in a closed domain:

$$I(f) = \int_a^b f(x) \mathrm{d}x.$$

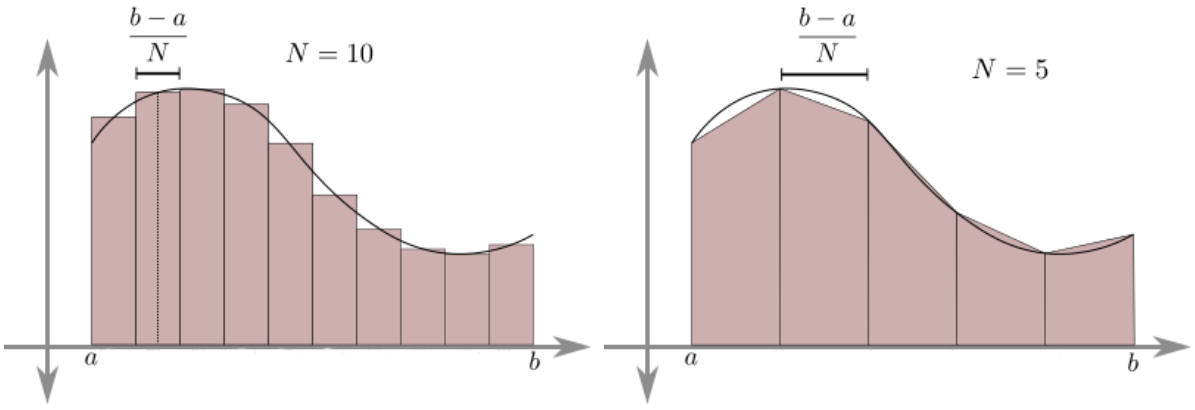
To do this, we investigate different quadrature rules I_Q that approximate I:

$$I(f) \approx I_Q(f) = \sum_{i=1}^{n} w_i f(x_i). \tag{1}$$

In other words, we approximate an integral by a sum of function values at given points x_i weighted by $w_i \in \mathbb{R}$.

Two well-known quadrature rules are the midpoint rule I_{mp} and the trapezoidal rule I_{TS} defined as follows:

$$I_{mp}(f) = \frac{b-a}{n} \sum_{i=0}^{n-1} f\left(a + \left(i + \frac{1}{2}\right) \frac{b-a}{n}\right),$$
$$I_{TS}(f) = \frac{b-a}{n} \left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f\left(a + i\frac{b-a}{n}\right)\right).$$



(a) Midpoint rule $I_{mp}(f)$ for $n = 10$

(b) Trapezoidal rule $I_{TS}(f)$ for $n = 5$

Take a look at the files `Midpoint.py` and `Trapezoidal.py`, where you will find a class that implements each of the quadrature rules.

1. Familiarize yourself with the code. Do you understand what the functions do?
2. Create a separate Python file and import the two classes. Create one object for each class and compute the approximate integral of a sine function in the domain $[0, 0.5]$ using $n = 100$ points.

Notice that the two classes are quite similar. They only differ in the function `constructMethod`. In other words, there's a lot of *code duplication*, which is not good. We can use the concept of *inheritance* to encapsulate their common functionalities. (In the end, they are both quadrature rules of the form (1)!)

Images: http://calculus.seas.upenn.edu/?n=Main.DiscreteIntegration

1. Write a new class called `QuadratureRule` that has the same three functions as `Midpoint` and `Trapezoidal`. The function `constructMethod` should stay unimplemented (using the keyword `pass`).

2. Rewrite the classes `Midpoint` and `Trapezoidal` so that they inherit from `QuadratureRule`. What can you get rid of?

3. Test your new classes to make sure they still work properly.

4. Create a new class that implements the Monte Carlo quadrature rule given by

$$I_{\mathrm{MC}}(f) = \frac{b-a}{n} \sum_{i=1}^{n} f(a + \xi_i (b-a)), \tag{2}$$

   where $\xi_i \in [0, 1)$ is a uniformly distributed random number. You can use the function `random.uniform(a,b)` to implement this rule.