EPFL Fall 2022
Doctoral School
Civil and Environmental Engineering

Scientific Programming
for Engineers

Week 14 — *Homework - Pybind and Trajectory Optimization*

The goal of this exercise is to use the external library Pybind11 to create Python bindings of the C++ Particles' code. The following points will be considered for the grading:

- Proper usage of git (meaningful comments, several commits with developments steps, use of .gitignore)
- Code works as intended
- Readability of the code (meaningful variable names, comments)
- Minimal documentation: README file

Use the starting point from GIT for this exercise

First part - Pybind: Python bindings for Particles Code

We have provided a **main.py** in the starting point. This script shows all the C++ classes and their respective functions to be exposed to Python. The objective of the exercise will be to ensure that the script works for all types of particle : Planet, PingPong and MaterialPoint.

Exercise 1: ***Factory Interface***

1. Create python bindings for all factory interface classes : `ParticlesFactory`, `MaterialPointsFactory`, `PlanetsFactory` and `PingPongBallsFactory`.
2. In class `ParticlesFactory`, `createSimulation` function has been overloaded to take functor as one of its argument. Comment on what this function is doing?
3. Create python binding for `createSimulation` function. You will have to use `overload_cast`. For more help, please refer : Overloading.

Exercise 2: ***Compute***

1. Create python binding for classes: `Compute`, `ComputeInteraction`, `ComputeGravity`, `ComputeTemperature` and `ComputeVerletIntegration`
2. How will you ensure that references to Compute objects type are correctly managed in the python bindings?
3. Some of the private members of class `ComputeTemperature` are made accessible in **main.py**. Create python bindings to access these variables and set their values.

Exercise 3: ***Other Classes***

1. Create python bindings for other necessary classes and their respective functions according to **main.py**.

Second part: Particle trajectory optimization

You will now use the python interface to simulate the motion of planets of our solar system.

Exercise 4: ***Units of the code***

1. In the file *init.csv* is stored the state of the planets at the first of January 2000. The coherent set of units is the following:
   - distance: AU= 149597870.700 km
   - time: 1 day
   - mass: $m_e = 5.97219e24$ kg where m_e is the mass of earth.
2. Use the starting point of the trajectory for the particle program and use it to simulate 365 days with a timestep of 1 day.
3. In the directory *trajectories* are stored files such that each file (one per step) contains the information of all the planets (named *step-XXXX.csv*). These trajectories are "measures" starting at the first of January 2000.
4. Verify that the dynamics correspond to what you expect but for Mercury. Indeed, an error in the calculation of the velocity has been made for Mercury so that the amplitude of this velocity is wrong (but not its direction). What follows aims at correcting this initial velocity.

Exercise 5: ***Compute the error of the simulation***

The goal of the present exercise is to calculate the integral error for

$$E(p) = \sqrt{\sum_{i=0}^{365} \mid \boldsymbol{X}_i^{ref,p} - \boldsymbol{X}_i^{p} \mid^2} \tag{1}$$

where p is a given planet, $\boldsymbol{X}_i^{ref,p}$ is the reference position of planet p at time i and $\boldsymbol{X}_i^{p}$ is the computed position of planet p at time i.

- Make a python function that reads the trajectory of a given planet and make a numpy array out of it.

  ```python
  def readPositions(planet_name,directory):
  ```

  The numpy to be returned should be a $m \times n$ matrix with $m = 365$ and $n = 3$, the columns being the three components of the planet position.

- Make function that compute the above mentioned error out of two numpy trajectories.

  ```python
  def computeError(positions,positions_ref)
  ```

  Test that function on the trajectory of Mercury

Exercise 6: ***Launching the particle code from python by generating the input***

- Make a python function that generates an input file from a given input file but by scaling velocity of a given planet

  ```python
  def generateInput(scale,planet_name,input_filename,output_filename)
  ```

- Make a function that launches the particle code on an provided input

  ```python
  def launchParticles(input,nb_steps,freq)
  ```

- By using all the previously mentioned functions make a function that gives the error for a given scaling velocity factor of a given planet

  ```
  def runAndComputeError(scale,planet_name,input,nb_steps,freq)
  ```

Exercise 7: ***Optimization***

- Use all the previously defined functions and the routine

  ```
  scipy.optimize.fmin
  ```

  to find the correct initial velocity for Mercury. Plot the evolution of the error versus the scaling factor. Describe in your README how to execute the optimization routine.