Assignment 3

Technical assignment 3: numpy and swig

PHY410: Do problems 1 and 2.

PHY 505: Do all three problems.

Instructions

1. Accept the assignment from GitHub Classroom: https://classroom.github.com/a/J2TJpGOL. GitHub will create a forked repository for you under github.com/ubsuny/phy410-assignment3-(your_username).
   - Note: replace (your_username) with your actual GitHub username
   - This repository is just a copy of github.com/ubsuny/CompPhys, our general repository for the class.
2. Clone the repository to your laptop. Inside, you'll find a folder `Assignment3`. Do all of your work inside this folder.

   ```
   cd PHY410 # Or whatever folder you use for this class
   git clone github.com/ubsuny/phy410-assignment3-<your_username>
   cd phy410-assignment3-<your_username>/Assignment3
   # Do the assignment here
   ```
3. Do the assignment. There is a separate jupyter notebook for each problem; do your work inside these notebooks, following the directions closely.
4. Once finished, you should submit two things:
   1. Submit your code to GitHub Classroom. To do this, do the usual `git add/commit/push`. For example,

      ```
      cd Assignment3
      git add ./*ipynb
      git add swig_code/*
      git commit -m "I hope I passed"
      git push origin main
      ```
   2. Submit your writeup, including a link to your github classroom area where your code is, on UBLearns.

The problems are described in this README file, but the full problem statements are contained in the Jupyter notebooks.

Problem 1: no loops!

(25 points)

First, we create $N = 10^6$ randomly distributed vectors of five numbers: $\vec{x}_i$ (where the index i ranges from 0 to $N-1$, and each vector $\vec{x}_i$ has length 5). Specifically, this is a numpy array with shape (N, 5).

Problem 1a

(15 points)

Without using explicit loops, compute the weighted average of each of the five numbers with (non-normalized) weights $\vec{W} = (5, 15, 30, 100, 400)$. That is, compute the quantity:

$$\frac{1}{N} \sum_{i=0}^{N-1} \vec{x}_i \cdot \vec{w}$$

where $\vec{w}$ represents the **normalized** weights, $\vec{w} = \vec{W} / \sum_{i=0}^{4} W_i$.

You are allowed and encouraged to use any numpy function, including average and mean.

Problem 1b

(10 points)

Calculate the same average as in Part (a), except only including vectors $\vec{x}_i$ for which the third element (index 2) is larger than the second element (index 1).

Problem 2: projectile motion

The equation for a projectile, i.e., an object undergoing constant acceleration due to gravity, is

$\vec{x}(t) = \frac{1}{2}\vec{a}_0 t^2 + \vec{v}_0 t + \vec{x}_0$

This problem uses numpy to study the motion of the projectile. Assume the motion is in 2 dimensions, x and y, with x being horizontal and y being vertical. We will create numpy arrays that represent the motion of the projectile at discrete time intervals. In particular,

- The time variable t should vary from 0 to 0.5 seconds, with a time step of 0.001 s.
- The initial position is $\vec{x}_0 = (0, 1)$ (units: m).
- The initial velocity is $\vec{v}_0 = (10, 0)$ (units: m/s).
- Gravity is $\vec{a}_0 = (0, g)$, where g varies depending on the scenario.

Problem 2a

(15 points)

Compute $\vec{x}(t)$ for Earth and Mars. Then, plot the x and y trajectories (i.e y vs x) using `matplotlib`, plotting Earth and Mars on the same plot. Label your axes and draw a legend. To get full credit, you must implement this without `for` loops in python. If you cannot manage without loops, that's good for partial credit (losing 5 points).

Problem 2b

(10 points)

Using array programming (i.e., **not** just calculating the solution of the quadratic equation, which is very easy), determine the approximate t and x where the projectile crosses the x axis, i.e., where $y = 0$. Specifically, using the numpy arrays from the previous step, determine the first x_i and t_i in the arrays representing x and t for which $y_i < 0$.

Problem 3:

In this exercise, we will create C++ classes for 2- and 3-vectors, similar to what we used in Assignment 2, Problem 2, and then load them into python using swig.

A full example is provided for `Vector2`, a class for a 2-dimensional vector (x, y).

We will repeat the steps above for a `Vector3` class, a 3-dimensional vector containing 3 doubles, (x, y, z). In 3(a), you will create all of the necessary files for the `Vector3` C++ class and its swig implementation. Then, in 3(b), you will define C++ functions and the swig implementation for the dot product and cross product of two `Vector3` objects.

Problem 3a

(15 points)

Following the example above for `Vector2`, create a `Vector3` class in C++ and a swig implementation. Specifically, you will need four files:

- `swig_code/Vector3.h`
- `swig_code/Vector3.cpp`
- `swig_code/vector3.i`
- `swig_code/setup_vector3.py`

To demonstrate that your code works, just like above, create two Vector3 objects, `v1 = (1, 2, 3)` and `v2 = (4, 5, 6)`, and a third vector `v3 = v1 + v2`. Print out the value of `v3`.

Problem 3b

(10 points)

Implement C++ functions that compute the dot product and cross product of two `Vector3` objects, plus the swig implementation both functions in a python module named `vector3_ops`. Specifically, create the following four files:

- Declare the functions in a header file, `swig_code/vector3_ops.h`. The functions should have the following signatures:
  - `Vector3& cross(const Vector3& v1, const Vector3& v2)`
  - `double dot(const Vector3& v1, const Vector3& v2)`
- Define the functions in a source file, `swig_code/vector3_ops.cpp`.
- Write a swig configuration file at `swig_code/vector3_ops.i`.
  - Note: you will need to include `vector3.i` (for reference, see `CompPhys/SwigExamples/swig_example/exampl` where we include std_vector.i).
- Write a python setup file at `swig_code/setup_vector3_ops.py`.
  - Note: in `sources`, you will need to include the C++ files for both `Vector3` and `vector3_ops`.

To demonstrate that your code works, execute the following: - Let `v1 = (1, 2, 3)` and `v2 = (4, 5, 6)` as above. - Print the dot product and the cross product of `v1 = (1, 2, 3)` and `v2 = (4, 5, 6)` defined above.