Module 4: N-body Simulation

JHU EP 606.206 - Introduction to Programming Using Python

Introduction

In this assignment you will simulate the motion of celestial bodies. Your code will model the gravitational forces between two bodies using classical kinematic equations. You'll use Python lists to store data for multiple planets and compute calculations at each step within the simulation.

Skills: lists, math/string functions, control structures (while loops), math operators

N-body Simulation

Recall Newton's law of universal gravitation (where G = 6.67E-11 N m^2 kg^{-2}):

$$F = \frac{G \times (m_1 \times m_2)}{r^2}$$

That is, the pairwise force between two bodies is proportional to the product of their masses divided by their distance squared times the gravitational constant.

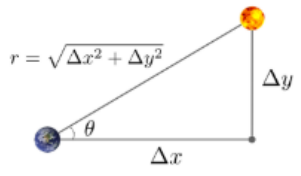
Our simulation will be in a 2D plane so the total force can be decomposed into x and y components:

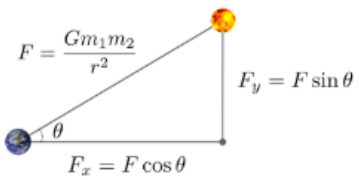
$F_x = F \times (\frac{\triangle(x)}{r})$	$F_y = F \times (\frac{\triangle(y)}{r})$

Additionally, Newton's 2nd Law states the acceleration of a body is the total force exerted on it divided by its mass. This can also be broken down into x and y components in a 2D coordinate plane:

$a_x = \frac{F_x}{m}$	$a_y = \frac{F_y}{m}$

Here is a diagram that demonstrates these principles (image courtesy of Princeton University CS):





$$\cos\theta = \frac{\Delta x}{r}, \quad \sin\theta = \frac{\Delta y}{r}$$

Input

You are given the total time simulation (t) in seconds, the time step in seconds, followed by (in order) the x position, y position, x velocity, y velocity, and mass in a Python list for the Earth, Mars, Mercury, the Sun, and Venus. You may copy and paste the lines below directly into your program (you do not have to read them in from the user). You may find it helpful to add each of these lists to a single list, maybe called 'planets', to form a nested list, but it certainly is not mandatory.

```
t = 157788000 # total time of simulation

dt = 25000 # time delta

earth = [1.4960e+11, 0.0000e+00, 0.0000e+00, 2.9800e+04, 5.9740e+24]

mars = [2.2790e+11, 0.0000e+00, 0.0000e+00, 2.4100e+04, 6.4190e+23]

mercury = [5.7900e+10, 0.0000e+00, 0.0000e+00, 4.7900e+04, 3.3020e+23]

sun = [0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 1.9890e+30]

venus = [1.0820e+11, 0.0000e+00, 0.0000e+00, 3.5000e+04, 4.8690e+24]
```

Physics Calculations

You will be given two parameters: a timestep and a total simulation time. We will recalculate the position at each timestep under the total amount of simulation time has elapsed. For example, if the simulation is 10 seconds long and the timestep is 0.5 seconds, your simulation will calculate the updated position 20 times. To calculate the updated position of two celestial bodies at a particular timestep:

1. Calculate the total force between the two bodies using Newton's Gravitational Law
    a. **Note**: When calculating the force you'll need to use the x- and y-positions of the Sun to calculate the force. In this one particular dataset, yes, the Sun is always centered at 0.0. However, we want you to start thinking about how to write code that is generic for potentially many different inputs (ones where the Sun may not be centered at the origin, for example). So, please avoid the temptation to hard-code the value 0.0 for the Sun's position when computing 'r' as part of your force calculation.
2. Use the result from (1) to compute the x and y components of the total force
3. Use the result from (2) to compute acceleration in the x and y direction
4. Use the results from (3) to calculate velocity in the x and y direction
    a. Velocity in a given direction is the current velocity in that direction plus the newly calculated acceleration in that direction multiplied by the timestep
5. Use the results from (4) to calculate the new x and y positions
    a. Position in a given direction is the current position in that direction plus the newly calculated velocity in that direction multiplied by the timestep

N-body Simulation

Your code will be in a file called **nbody.py** and will operate on the data provided in the Input section above. You will create three variables: t = total time of the simulation, dt = the length of the timestep, and t_total will be a running total of the amount of time that has elapsed (t and dt will be given to you). In a loop that executes while t_total < t you will perform the calculations from the 'Physics Calculations' section. When the loop is complete you will be left with the final position, at time t, of all N planets from the input! The output format is explained in more detail in the Outputs section. Here is an example where t = 50000 and dt = 25000 for the input file in the 'Input File Format' section:

```
1.4959e+11 1.4900e+09 -2.9639e+02 2.9799e+04 5.9740e+24
2.2790e+11 1.2050e+09 -1.2772e+02 2.4100e+04 6.4190e+23
5.7826e+10 2.3945e+09 -1.9789e+03 4.7880e+04 3.3020e+23
0.0000e+00 0.0000e+00 0.0000e+00 0.0000e+00 1.9890e+30
1.0818e+11 1.7499e+09 -5.6661e+02 3.4998e+04 4.8690e+24
```

Pseudocode

To help you get started we wanted to introduce you to a new, extremely helpful for helping you make the transition from understanding a problem conceptually to creating a solution through code. This technique is known as **pseudocode**. Pseudocode allows you to translate a problem into a very loosely-defined intermediate representation that is a syntax-less mix of words and code. You might not have realized it, but we actually provided you with pseudocode for the Module 3 Assignment when we provided the 5-step outline of how Luhn's verify/generation algorithm work!

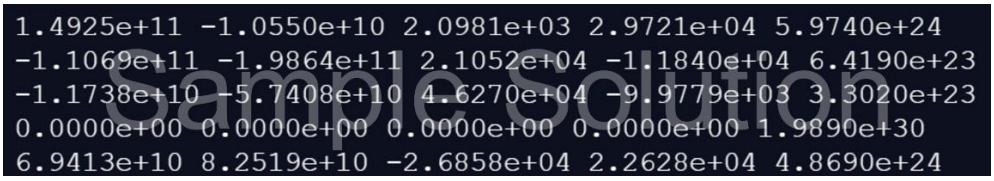
Pseudocode doesn't have to consist solely of words. It can be any mix plain English and code that helps you get from the problem statement to your final solution. Below is some pseudocode to help you get started with the Module 4 Assignment. Before you take a look at it, though, please keep a few important things in mind:

1. **The provided pseudocode is NOT intended to be a 100% solution**; we're providing it as a means to help you get started if you're unsure where to begin. It'll probably get you 80-90% of the way to a solution, but you are NOT obligated to take the approach shown by any means. And please
2. **Please do not depend on pseudocode being made available for all Assignments in this course**. We will provide varying levels of pseudocode for a few Assignments but will gently scale back the amount provided as we progress. In later Assignments where no pseudocode is provided, it may be helpful for you to begin by the Assignment by writing it on your own.
3. **The pseudocode is NOT meant to be copy-pasted**. Doing so will do nothing to help you learn to implement solutions organically and, as mentioned in #2, it won't be there in future Assignments/classes, nor in the workplace. At a minimum, please make an effort to at least transcribe it manually.

```
import math

N = 5
G = 6.67E-11
px = []
py = []
vx = []
vy = []
m = []

SUN = 3

# created a nested list containing the input planet data from above

t_total = 0.0
dt = 25000.0 # delta t
t = 157788000 # total simulation time

while t_total < t:
    for i in range(N):

        # skip the sun

        # calculate the radius between i'th planet and the sun

        # calculate the pair-wise force between i'th planet and the sun

        # calculate the x and y components of the force

        # calculate the x and y components of the acceleration
        # for the current timestep

        # calculate the x and y components of the velocity
        # for the next time step

        # calculate the x and y components of the resulting position

    # update the time by delta_t

# output the formatted values for the x and y components of
position/velocity and mass
for i in range(N):
    # print formatted output for each planet
```

Outputs

Your output should contain each field, rounded to 4 decimal digits, separated by a single space, for each of the N bodies provided in the input (in order) at the end of the simulation. Take a look at the "Python f-string format floats" section of the this article to see an example of using f-strings to specify the number of decimal places in a floating-point number. Below is a screenshot of my final solution for reference:



Deliverables

readme.txt

So-called "read me" files are a common way for developers to leave high-level notes about their applications. Here's an example of a README file for the Apache Spark project. They usually contain details about required software versions, installation instructions, contact information, etc. For our purposes, your readme.txt file will be a way for you to describe the approach you took to complete the assignment so that, in the event you may not quite get your solution working correctly, we can still award credit based on what you were trying to do. Think of it as the verbalization of what your code does (or is supposed to do). Your readme.txt file should contain the following:

1. **Name**: Your name and JHED ID
2. **Module Info**: The Module name/number along with the title of the assignment and its due date
3. **Approach**: a detailed description of the approach you implemented to solving the assignment. Be as specific as possible. If you are sorting a list of 2D points in a plane, describe the class you used to represent a point, the data structures you used to store them, and the algorithm you used to sort them, for example. The more descriptive you are, the more credit we can award in the event your solution doesn't fully work.
4. **Known Bugs**: describe the areas, if any, where your code has any known bugs. If you're asked to write a function to do a computation but you know your function returns an incorrect result, this should be noted here. Please also state how you would go about fixing the bug. If your code produces results correctly you do not have to include this section.

Please submit your ***nbody.py*** source code files along with a PDF file containing screenshots of your output in a PDF called JHEDID_mod4.pdf (ex: jkovba1_mod4.pdf). Please do not ZIP your files together.

Recap:

1. readme.txt
2. nbody.py
3. 1 screenshot of your output from nbody.py file

Please let us know if you have any questions via Teams or email!