Satellite project, AST 2000

Part 2: Calculating planetary orbits

In this part the goal is to calculate and make a list of the positions of all the planets at each timestep for a long period of time. You can then use this list as a reference when you later need to know the position of the planets at some particuar time. This list is all you will need for the rest of the project from this part. You will probably not use the code again.

This part is similar to exercise 1B.7 in part1B (except the part on the 3-body problem which you do not need here). Please read through that exercise before you read on.

In order to simplify this problem we make a few assumptions:

- We neglect the gravitational force between the planets.
- We neglect the force on the star from the planets and keep the star fixed at the origin. (in an exercise further down you will relax this assumption, but only for that exercise. For all other calculations during this project work, the assumption of the star fixed to the origin is necessary to make the communication with the SolarSystem class correct)
- All planetary orbits are in the same plane (defined to be the x-y plane).
- All planets orbit *and* self-rotate counterclockwise (right hand system).

The star is assumed stationary at the origin of our coordinate system, and all the planets orbit independently of each other.

For each of the planets you need to solve the following differential equation, given by Newton's second law

$$m_P \frac{d^2\mathbf{r}}{dt^2} = \mathbf{F_G} = -\frac{Gm_P m_S}{r^3}\mathbf{r}, \tag{1}$$

where m_P is the mass of the planet, m_S is the mass of the star, $\mathbf{r}$ is the position vector of the planet and $r = |\mathbf{r}| = \sqrt{x^2 + y^2}$ is the distance between the star and the planet. Use the Euler-Cromer method to solve this ordinary differential equation set numerically for each planet.

Hints :

- Write the second order differential equation (eqn 1) into a set of first order differential equations for the variables $\mathbf{r}$ and $\mathbf{v}$.

- Use smart units! We recommend AU's (astronomical units) for length, years for time, and solar masses, $M_\odot$, for mass. In these units the gravitational constant is just given by

$$G = 4\pi^2[\mathrm{AU}^3\mathrm{yr}^{-2}\mathrm{M}_\odot^{-1}].$$

Optional exercise: Can you show why $G = 4\pi^2$?

- Use at least 10 000 timesteps per year (some small/fast systems will need up to 100 000), and run the simulation for at least 20 orbits of your home planet. This sounds like alot, but we will need the positions of the planets with very good accuracy later. This should also be a motivation to optimize your code: try to find ways to make it faster.

- Solve the problem for one planet first and make a plot of the orbit to make sure your code works.

- Use numpy arrays! They have many useful properties (like elementwise operations, splitting etc.). Another useful property for this part is that you can easily save a numpy array directly to file. Much of the data that you will recieve later will be given in terms of saved numpy arrays, so learn how to handle these. This makes it very easy to use the results from this part later. Here is an example:

```
    import numpy as np
    # File for saving positions.
    outFile = open('positionFile.npy','wb')
    ...
    #Calculate results
    #Generate numpy array with positions and
    #a corresponding array of times.
    ...
    #Save combined array to file
    np.save(outFile, [positions,timeArray])
    outFile.close()
```

  And if we want to use the arrays in another code

```
    import numpy as np
    # File with saved positions.
    inFile = open('positionFile.npy','rb')
    # Put positions and times into new arrays
    [planetPositions, times] = np.load(inFile)
    inFile.close()
```

- The choice of integration methods is free, and in many cases it is sufficient to stick to Euler-Cromer. However, there exists an integration method called Leapfrog which is slightly more precise than Euler-Cromer, with the same amount of computations per time step. For forces that only depend on position, the Leapfrog method is actually second order in Δt, and conserves mechanical energy during orbital motion avoiding numerical spiral motion. If you want to try it out, you can probably save some computing time without having to implement and debug e.g. a Runge-Kutta 4. As the name implies, the Leapfrog calculates the velocities of the particles at half-steps between the position calculations. You start with the position at $t = 0$, use the acceleration there to calculate the velocity at $t = dt/2$, you then use this velocity to leap-frog the position from $t = 0$ to $t = dt$, calculate the acceleration at $t = dt$ and use it to leap-frog the velocity from $t = dt/2$ to $t = dt \times 3/2$, and so on. The only thing you need to do to convert an Euler-integrator to a Leapfrog integrator, is to add a half-step to the initial velocity before the time loop, such that you start with $v_{1/2} = v_0 + a_0 * dt/2$. For each step in the time loop, you first calculate the new position from the half-step velocity, then the acceleration at the new position, and finally the next half-step velocity from the acceleration.

Part 2.1: Visualizing and checking the orbit of the planets

In order to visualize and check your orbits you can dump your position- and time arrays to an xml-file and use the SolarSystem Viewer exactly as described in exercise 1B.7 by using the `orbit_xml` method.

```
#Calculate positions of each planet for each time step.
#Must fit into array with shape [2, N_planets, N_time]

myStarSystem.orbit_xml(positions, times) #make an xml

#If you have many time steps, you can slice every 100th.
myStarSystem.orbit_xml(positions[:,:,::100], times[::100])
```

Finally, when your positions are calculated and appear correct in SolarSystem Viewer, you can check whether your orbital calculations are sufficiently exact using the built-in method `check_planet_positions` in the AST2000SolarSystem class. The class takes 3 arguments: planet positions, years of simulation and number of time steps pr. year (in that order). You may enconter numerical problems or other kinds of problems later in the project if your orbital calculations are not sufficienly exact. If you are within a 1% margin the module will allow you to continue to the next part. If not, try smaller time steps and check your calculations again. This test has to be passed to be able to move on.

When you pass this test, a npy file called `planet_positions.npy` will be created. This npy file contains a numpy array with the exact positions of your planets with the time resolution you entered (a new time array is also included). You can use these positions further on in your project to precisely hit your target planet. If you do not use these precise position, you also risk launching your satellite from the wrong coordinates (this will not be the case if you launch at $t = 0$, but if you wait a while before you launch your launch position could well be at the moon if you're really unlucky. If you launch at $t = 0$ you can therefor disregard this npy file.). The reason for this is that even if you make very tiny time steps, there will always be some numerical errors present.

Part 2.2: Testing your orbits using Kepler's laws

This section is optional for those working on the project alone. Those working in groups need to do this part. Now we can check the validity of our orbital calculations using Kepler's laws. We know from Newton's laws that they need to be valid.

1. Start by deducing Kepler's 2nd and 3rd law following exercise 1B.1 and 1B.2.
2. Now choose the planet of your solar system with the largest eccentricity e (remember: you can get this number directly from AST2000SolarSystem class).
3. Plot your numerical orbit for this planet and then plot the analytical orbit on top using eq.13 from lecture notes 1B. How well do they fit?
4. Now we want to integrate the area swept out by the position vector as in Kepler's 2nd law. Choose a starting point close to the aphelion of the orbit. Choose a suitable time period Δt when the planet is always in the vicinity of aphelion. Integrate the area swept out by the position vector from the starting point and during a time Δt. You may use some hints from exercise 1B.1 to do the numerical integration (it should be straight forward).
5. Now do the same integration during the same period of time Δt, but now when the planet is close to perihelion. To which precision are the two areas equal?
6. How large distances did the planet travel in each of the two cases?
7. What was the mean velocity of the planet during these two periods?
8. Now check if Kepler's 3rd law is valid for all your orbits, by taking the ratio P^2/a^3

Part 2.3: Can an extraterrestrial discover your solar system?

In this exercise you will check whether extraterrestrials in a solar system far away will be able to discover that there are planets orbiting your star by looking at its velocity curve. And indeed you will choose some extraterrestrials to analyze your data.

1. Read through exercise 1C.5 which you shall now do, but choosing **only one of the planets in your system, one of the largest planets and preferably as close to the star as possible**. It is important that you for the moment only use one planet in solving 1C.5. If you are working in a group you should do all of 1C.5, if you are working alone it is sufficient to do (1) and (2) (omitting (3) and (4)).
2. You should now do exercise 1B.4. Then calculate the total energy of your two-body system through one full period. Is the energy conserved? This is a test of your numerical orbital calculations.
3. From 1C.5 you should now have a noisy velocity curve of your star (and possibly a light curve). Now contact another student/group following the project and switch velocity/(light) curves.

4. Read through exercise 1C.4 and use this exercise as an aid to estimate the mass of the planet of the other student using the data you switched. If you work in groups and also have light curves, check if the quality of the light curves are good enough in order to estimate also the radius and density of the planet.

5. After you have both estimated the mass of the planet of the other student/group, exchange the actual masses and check how well they fitted.

6. **If you work in groups:** Make the velocity curve of the star (and only the velocity curve) taking into account the 3-4 largest planets of your system. Agin exchange velocity curves and see if the other group can guess how many planets you included.

It is important to note that the planetary orbits that you calculate in this exercise, where the motion of the star is taken into account, should **not** be used further in this project. In the rest of this project you must use the assumption of the star fixed at the origin, as you did above, in order to communicate correctly with the SolarSystem class.