The orbits of a planet and a binary star system

These instructions are modifications of Problems 3.P.76 and 3.P.77 (page 135). In Problem 3.P.76 you predict the motion of a planet around a nearly stationary star. In Problem 3.P.77 you predict the motion of a binary star system, in which two stars orbit each other.

You may find it useful to refer to your fan cart program for the basic structure of your new program, but it is simpler and clearer to start a new program rather than trying to modify the fan cart program. The main difference is that the net force on the fan cart was constant, but the gravitational force can vary in both magnitude and direction at different locations, as you saw in your program that calculated the gravitational forces acting between the asteroid Mathilde and a passing spacecraft. You may wish to refer to your Mathilde program to remind yourself of the steps involved in calculating gravitational force as a vector.

1 Creating the objects

Actual data for the Sun and Earth may be found on the inside back cover of the textbook. We'll make the radii of the Earth and Sun much bigger than they really are in order to be able to see them in the vast reaches of space! If we did not exaggerate the size of Sun and Earth, we simply couldn't see them. Try this humbling exercise... we are certainly of insignificant size, compared to the distances between planets! the Place the Earth at its normal distance from the Sun, to the right of the Sun (along the x axis):

```
from visual import *
Sun = sphere(pos=vector(0,0,0), radius=1e10, color=color.yellow,
make_trail=True)
Earth = sphere(pos=vector(1.5e11,0,0), radius=5e9, color=color.cyan,
make_trail=True)
scene.autoscale = 0
```

What is that last statement? Normally, VPython automatically zooms out if the scene gets bigger and zooms in if the scene gets smaller, so that you can always see the entire scene. The statement **`scene.autoscale = 0`** says "turn off autoscaling for this scene". Having autoscaling off makes it easier to view some of the orbits you are going to display.

- Run the program to see the Sun and Earth.

2 Giving the objects physical attributes

In addition to geometrical attributes such as **`pos`** and **`radius`**, you can give objects physical attributes such as mass. Give the Sun and Earth the correct masses (see inside back cover of textbook). Also, define the gravitational constant *G:*

```
G = ?
Sun.m = ?
Earth.m = ?
```

3 Specifying initial conditions

You've already specified the initial positions of the Sun and Earth. You also need to specify their initial momenta. Because it has such huge mass, we will assume the Sun hardly moves, and its velocity is initially zero (actually, it is drifting through space). Suppose the Earth is initially headed in the +*y* direction with speed 2e4 m/s (note that this is *not* the momentum!). Specify the initial momenta, as vectors.

```
Sun.p = ?
Earth.p = ?
Earth.trail = curve(color=Earth.color)
```

Also choose a value for Δt, the time interval you will use in updating momentum and position. The criteria are that Δt should be small enough that neither the force nor the velocity are changing very much (so that the updates are accurate), yet Δt should be big enough that you're not waiting forever for the computer to calculate a huge number of updates. Think about what you know about how long it takes the Earth to make a complete circular orbit of the Sun, and the criteria just mentioned, and choose a reasonable value for Δt. Also, set a clock time to 0 to be able to time how long an orbit takes.

```
deltat = ?
t = 0
```

4 The while loop

Again thinking about what you know about how long it takes the Earth to go around the Sun (remember that the units we are using are seconds, not days or years), choose a suitable stopping time for your **`while`** loop:

```
while t < ?:
```

Inside your while loop you need to do the following:

- Calculate the gravitational force acting on the Earth.
- Update the momentum of the Earth.
- Update the position of the Earth.
- Update the time *t*.

In your calculations, use the position of the Sun in the form **`Sun.pos`** rather than assuming it is always at the origin. This will make it easier to write a binary star program later, when both stars will move.

- Run the program.
- If it runs too fast, slow it down by using a rate statement, such as **`rate(300)`**.
- If it runs too slowly, increase Δt.
- Adjust the stopping time so that the Earth makes several complete orbits.

5 Accuracy

If you use a very large Δt, the calculation is inaccurate, because during that large time interval the force changes a lot, making the momentum update inaccurate, and the momentum changes a lot, making the position update inaccurate. On the other hand, if you use a very small Δt for high accuracy, the program runs very slowly. In computational science, there is a trade-off between accuracy and speed.

How can you tell whether an orbit is accurate? There's a simple test: Make Δt smaller and see whether the motion changes. That is, see whether the orbit changes shape. (Obviously the program will run more slowly.)

- Experiment with your value for Δt, to find a good compromise between accuracy and speed. Cut the step size (Δt) and see whether the motion changes or not. Let the Earth go around the Sun five times in this test.
- To see the effects of Δt being too large, increase Δt until the motion is different than it is with small Δt.
- **In your notes, record the (approximate) biggest value of Δt you can use and still get accurate motion in five trips around the Sun. You will be asked for this value when you turn in your program.**
- I will ask you to demonstrate an accurate orbit and a deliberately inaccurate orbit.

6 Visualizing the momentum vector with an arrow

Next you will visualize the Earth's momentum by creating an arrow before the while loop and updating it inside the loop. You need to know the approximate magnitude of the momentum in order to be able to scale the arrow to fit into the scene, so inside the loop, print the momentum vector:

```
print (“p=”, Earth.p)
```

As an example, suppose the magnitude of the momentum were about 1e31kg·m/s, and you want to scale down to 1e11 m. What would you need as a momentum scale factor **`pscale`** in this case? You would need a factor **`pscale`** that is approximately 1e11/1e31, so that if you multiply **`pscale`** times a momentum whose magnitude is about 1e31 kg·m/s, you get a length of about 1e31*(1e11/1e31) = 1e11.

Or suppose that the magnitude of the momentum were 100 kg·m/s; then you would need **`pscale`** to be about 1e11/100, so that if you multiply **`pscale`** times a momentum whose magnitude is about 100 kg·m/s, you get a length of about 100*(1e11/100) = 1e11.
Take a look at your print statement to see what is your specific case, in order to decide on the necessary scale factor for momentum.

Before the while loop, insert these statements:

```
pscale = ?
parr = arrow(color=color.red)
```

The last statement creates an arrow object with default position and axis attributes. You will set these attributes inside the loop.

- Comment out your **`print`** statement (that is, put a "#" sign in front of it), which slows down orbit plotting.
- Inside your loop, update the **`pos`** attribute of the **`parr`** arrow object to be on the Earth, and update its **`axis`** attribute to represent the current vector value of the momentum of the Earth (multipled by **`pscale`**).
- Run the program. Use a rate statement, or a small time interval (Δt) to slow the orbit enough to be able to see clearly the behavior of the momentum vector.

You may have to adjust the scale factor once you have seen the full orbit.

Questions your instructor will ask you at the next checkpoint:

- For this elliptical orbit, what is the direction of the momentum vector? Tangential? Radial? Something else?
- What happens to the momentum as it approaches the Sun? As it moves away from the Sun? Why? You should be able to explain this qualitatively in terms of the momentum principle.

7 Visualizing the force vector with an arrow

Using the same ideas you used to display the momentum as an arrow, display an arrow representing the force acting on the Earth.

- Display the force acting on the Earth, as an arrow with its tail always on the Earth.

Questions your instructor will ask you at the next checkpoint:

- For this elliptical orbit, is the force vector always perpendicular to the momentum vector?
- When is the gravitational force large? Small?

8 Other orbits

- Change the initial speed of the Earth to 3.5e4 m/s and see what kind of orbit you get.

You will have to zoom out to see the whole orbit (since autoscaling is turned off).

9 Circular orbits

The first part of Problem 3.P.76 asks you to set the initial conditions so as to get a circular orbit of our Earth around our Sun, to check that the program is working properly. Note that the initial speed of the Earth can be calculated from the distance it goes around the Sun in one year.

- Give this initial speed to the Earth and see whether you get a circular orbit.
- Change your loop to stop when t is the number of seconds in a year, and see whether in fact the Earth now takes one year to make one circular orbit.
- Reflection: What kind of orbit do you get if the initial speed is too slow for a circular orbit? Too fast?

You can now turn in your program. Be sure to check that the file you turn in is correct!

Continue to next page

10 A binary star

Problem 3.P.77 (page 135): About half of the visible “stars” are actually systems consisting of two stars orbiting each other, called “binary stars.”

- Change the mass of the “Earth” to be half the mass of our Sun.
- Give the “Earth” the initial velocity that our actual Earth has, and give the Sun an initial momentum **`Sun.p = vector(0,0,0)`**.
- Turn autoscaling back on by commenting out the statement **`scene.autoscale = 0`**.
- Inside your loop, comment out the arrow update statements.
- Inside your loop, update the Sun’s momentum and position, and update its trail. To update the Sun’s momentum, think carefully before doing lots more calculations: they aren’t needed!
- Run the program. What do you see?
- If you don’t have complete orbits, increase the time in your while statement.

- Experiment with other initial momenta for the Sun and “Earth”.
- In particular, try this: give the Sun an initial momentum that is equal in magnitude but opposite in direction to the initial momentum of the “Earth”. What is special about the orbit you observe? What is the total initial momentum? What does the momentum principle predict for the total momentum later? Explain.

You can now turn in your program. Be sure to check that the file you turn in is correct!

11 Playing around

What happens if you aim the objects straight away from each other? With large or small initial speeds?

What happens if you aim the objects straight toward each other? (When the objects get very close, the force changes rapidly with distance, so the calculations become increasingly inaccurate and the whole thing may seem to blow up, which is not what would actually happen.)

Our Sun is not actually stationary but is moving in a giant orbit around the center of our Milky Way galaxy (and the galaxy is moving toward the Andromeda galaxy.) Restore the Earth’s mass to its actual mass, and give it an initial velocity < 0, 0, 2e4 > m/s. Give the Sun an initial velocity < –100, 1000, 0 > m/s. Watch as the Earth orbits the moving Sun. What is the shape of the Earth’s curve?

You might like to add a third star to your binary star and see what wild kinds of orbits you can achieve. The orbits for a binary star can only be straight lines, circles, ellipses, parabolas, or hyperbolas. But with three bodies orbiting each other the orbits can be very diverse and very complicated. Except for some very special cases (very specially chosen initial conditions), the “three-body” problem has been shown to have no analytical solution (that is, there is no math function that gives the form of the orbits), but it is easy to instruct a computer to carry out a numerical solution as you did for the binary star. A caution: Update all the momenta BEFORE updating all the positions. Otherwise the calculations of gravitational forces would mix positions corresponding to different times.

You might like to model the Sun-Earth-Moon system, or the Solar System. A practical difficulty with visualizing the Sun-Earth-Moon system is that the Earth-Moon system is very small compared to the great distance to the Sun, so it’s just hard to see the details of the Moon’s motion.