

PHYSICS 113 – Recitation Assignment 3

Name

Recitation Assignment # 3 – The Two-Body Problem

October 7, 2009

Today’s recitation will involve the general form of gravity, and, in particular, you will show using a computer that two bodies will orbit one another in an elliptical orbit. Moreover, you will use numerics to determine the optimal speed for a particle traveling in a circular orbit. Note, for simplicity, when I use the symbol \odot , it means “of the sun”, and when I use \oplus , it means, “of the earth.”

You may complete this in class. However, if you are unable to do so, it is expected that you complete this for recitation next time. Remember to email both programs to Coleman when you are done, and be sure to put put: “Physics 113 yourname prog3.py” in the subject line.

You should also include a comment line in your code indicating whose program it is (i.e. yours).

It is also strongly recommended that you test your code after each step in the process.

If you have any questions, please ask.

New Concepts:

Physical

- Central gravity
- Two body interaction

Computational

- Nested Loops
- Lists of objects

Part I

In the first part of the assignment, we will model the earth-sun system. Since the sun is much more massive than the earth, we will be able to make the assumption that the sun is stationary.

1. Log into a workstation, pull up firefox, make yourself comfortable, etc.
2. In your “progs” directory (or whatever you’ve chosen to call it), create a new program using emacs called “prog3a.py”.
3. As with all of your python programs which use graphics, you will want:

```
from visual import *
```

at the top.

4. Create two spheres. Make one yellow, and at the origin, and the other blue, and at a position 1.5×10^{11} m from the central body, and along the x-axis.

Also add the following lines to your code:

```
scene.range=2e11
scene.autoscale=0
```

The first line makes the viewscreen large enough to see a bit beyond the “orbit” of the earth, and the second command tells the code not to automatically resize the picture if the earth flies out of the field of view.

Start by making the radii the true values from our solar system (look this up using google or in your textbook). Too small? Maybe you need to make them bigger.

5. Here’s where we start to add physics. The sun has a mass of 2×10^{30} kg, and the earth has a mass of 6×10^{24} kg. Set both of these. For example, the mass of the sun can be set by writing: `sun.m=2e30` and similarly for the earth.

Give the earth an initial velocity of 35 km/s=35000m/s in the y-direction. In general, you should be keeping track of the *momentum* not the velocity, so you should compute the corresponding momentum as follows

This can be done by entering:

```
earth.p=vector(0,35000*earth.m,0)
```

6. Remember the relation we discussed in class (in the non-relativistic limit):

$$\Delta \vec{r} = \vec{p}/m\Delta t$$

Well, at each timestep, you have to evolve the position of the earth, according to this formula. Put in the following additions to your code:

```
yr=3.15e7
dt=0.02*yr
t=0
```

```
while (t < 2*yr) :
    rate (10)
    earth.pos=earth.pos+dt*earth.p/earth.m
    t=t+dt
```

The first three commands tell the code how many seconds there are in a year. The second tells it what the value of Δt is. The third tells it to “start the clock” at $t=0$.

7. As we saw, the earth just flies in a straight line, when, in fact, we know that it moves around the sun. What makes it move? Gravity! The acceleration due to gravity can be expressed as:

$$\vec{F}_{\oplus} = -\frac{GM_{\odot}M_{\oplus}}{r_{sun-earth}^3}(\vec{r}_{\oplus} - \vec{r}_{\odot})$$

where $G = 6.67 \times 10^{-11} \frac{Nm^2}{kg^2}$ is the gravitational constant.

You should define the force on the earth as:

```
earth.F=-G*sun.m*earth.m ...
```

In order to compute the rest of the terms in the force, you may need to do a bit of searching on the vpython or python webpages, or using google.

Incidentally, the reason that the denominator has an r^3 (rather than r^2) is that the difference in vectors in the numerator divided by one power of r produces the unit vector, \hat{r} (think about it).

So, before the “while” loop, put in the definition of the gravitational constant. Within the loop, calculate the force, and update the momentum.

8. Put in a “curve” following the position of the position of the earth.
9. To prove you’ve mastered the code, do the following:
 - Adjust the initial velocity of the earth so that it moves more nearly in a circular orbit.
 - Run the code for 5 years rather than 2.
 - **Extra Credit:** Put Jupiter into your system. Find the mass, distance from the sun, and orbital velocity of Jupiter by using google.

Part II

In reality, it is cumbersome keeping track of a bunch of things called “earth”, “sun”, “jupiter”, etc. What if we were modeling 10 bodies, or 100? In this exercise, we’ll show how to use lists in order to make your code tidier, and easier to expand to more bodies.

10. Close your emacs window, and then copy your old code into a new file:

```
cp prog3a.py prog3b.py
```

Near the top of your code, enter:

```
star=[]
```

This says that you will create a list of objects called “star” which has no entries. In order to create stars, you need to do the following:

```
star.append(sphere(...))
```

These commands should *replace* the ones from your earlier program which say “earth=sphere(...)”, for example. Each time you do “append”, you add a star to the list.

Now, at all points in your program where you have “sun.p=..” for example, you can replace this with “star[0].p=...”

11. Make the mass of star “0” equal to 2 solar masses, and star “1” equal to 1 solar mass.
12. There is still something wrong with the code. Star “0” doesn’t move! This was an okay assumption when the earth was orbiting the sun because the ratios of their masses was so high. Now, however, you should make sure that the physics works the same for both stars. Verify your code is working by printing out the *total* momentum of the system at the end of each timestep.

13. Instead of writing explicitly, “star[0].p=star[0].p ...” and then the same for star[1], perhaps it would be better to do a loop. Wherever you are doing a calculation on both stars, you’ll want the following:

```
for mystar in (star):  
    mystar.pos=mystar.pos+mystar.p/star.m*dt
```

for example, along with any other calculations which need to be done on all stars. Note also that this loop should be “nested” (doubly indented) if it is under the time loop.

14. Make sure a yellow trail follows star “0” and a blue trail follows star “1”.