Week 8 Exercises

Physics 91SI Spring 2011 Handout 20

Alex Ji and Zahan Malkani

Planets Orbital Motion

For this week's exercise, we're going to make a simulation of planets orbiting the sun in the two-dimensional plane that we are most used to.

There are a few assumptions that we will make to simplify matters.

- The gravitational effect of the sun drowns out all other influences. Don't bother simulating any other interactions
- The two planets that we are chiefly concerned with are Venus and Earth. We can add more later, but it takes a couple of lines of code.

The Math Behind It All

The differential equation we will be using in the simulation is the straightforward one describing Newtonian gravity in two dimensions with it's inverse square law dependence.

$$\frac{d^2x}{dt^2} = \ddot{x} = \frac{\mathrm{M_{sun}}}{x^2+y^2}\cos\theta \tag{1}$$

$$\frac{d^2y}{dt^2} = \ddot{y} = \frac{\mathrm{M_{sun}}}{x^2+y^2}\sin\theta \tag{2}$$

θ is the angle between the planets precession and a arbitrarily decided upon horizontal axis.

$$\theta = \tan^{-1}\frac{y}{x} \tag{3}$$

This logic needs to be specified in the `orbital_motion` method. This method needs to return a list of the *differences* in each of the variables being modelled in the differential equation. The order in the list must match the order in which you access the variable passed in using the `y` variable, whcih must also be the same order that you use to specify initial conditions in `run_sim`. Probably best to specify an order in one big comment at the top and follow that throguhout!

Bu there is a problem, `scipy`'s `integrator` can only handle up a first order differential equation (obviously - since you are returning that very difference). So we'll use the trick that Jeff showed

us in class. Add variables that we model to our equation, namely if we are modelling x, we'll also model $\dot{x}$.

So to flesh this out further, if you pass in a list $[x, \dot{x}, y, \dot{y}]$ the first element of the list you return should be $[\dot{x}, \ddot{x}, \dot{y}, \ddot{y}]$. (Look at the differential equation to calculate those!)

With that much spelled out, go and implement `orbital_motion`! Look at the example of `basin.py` for help.

Initial Conditions

Use the constants defined at the bottom of the file to pick out initial conditions for any two planets. That is, you need to specify the location in x and y coordinates, along with each initial component of velocity.

Give this a bit of thought. I would also suggest starting by simulating Earth and Venus.

Plotting and Setting Up the Simulation

Next on your hitlist is to implement the `plot_positions` method. You are making a simple scatter plot (probably including a circle in the center for the sun).

You probably want the `x` and `y` parameters passed into `plot_positions` to be treated as lists of `(x,y)` pairs of positions at any given time for the sun and planets you are simulating.

The `savefig` line at the bottom just saves your masterful plot to a file in a folder called `data` (make sure that such a folder exists **before** running your simulation). We will be using these files to make our animation at the end.

The last method we need to add some logic to is `run_sim`. Choose `t_stop` and `dt` sensibly so that you don't have more than few hundred images. Pass positions appropriately to `plot_positions`.

You are now ready to actually run the simulation. If you feel a little unsure of what you've done ask one of us to give your a code a quick look (just because these simulations take a while to run).

Making the Animation

We'll be using a command line program `ffmpeg` to stitch the images together to make an animation. Incidentally `ffmpeg` is an industry-standard tool to convert video and audio formats, check it out if you have such a project.

Now, `cd` into the data directory and check that all the images have been saved correctly. Using `ffmpeg` for the video conversion

```
ffmpeg -r 10 -b 1800 -i sim%03d.jpg final_sim.mp4
```

Use `scp` or `http://afs.stanford.edu` to copy the video over to your local computer, and view your simulation in your favourite movie player! (Use VLC if you don't have a favourite.)

Further Steps

For an easy and cool addition, add another planet, or two? Simulate the inner solar system. In addition modifying `orbital_motion` you will also need to specify apropriate initial conditions. Remember though that the more planets you add, the longer the integrator will take, but it shouldn't take too much time.

A complicated addition is to also model the interactions *between* planets. Hint, all you need to change is `orbital_motion` method, but think through it. Use whiteboards if you want.

Once you're done, I'll have you know that in these two hour, you implemented what would be considered a rather advanced final project in other classes. Well done. ;)

Regular Expressions Exercises

Based on Software Carpentry exercises.

Parsing a file

In the repository, there should be a file called `people.txt`. Using regular expressions, write code that will read from the file, and print out the phone number for each individual. There's starter code in regex.py.

Then write another regular expression that just prints the area code.

Then write another regular expression that just prints the e-mail.

Parsing Time Formats

Write a function called time_match that takes in one parameter (a string). Use regular expressions to match the string with the following two time formats:

HH:MM:SS in 24-hour format. Here, we require two digits for the hour, minutes, and seconds.

HH:MM PM/AM in 12-hour format. Here, we require one or two digits for the hour, two digits for the minutes, and the seconds should not be included.

So, for example, "15:41:38" and "03:29:10" are valid matches for the first format, and "2:30 PM" and "10:54 AM" are valid matches for second format.

If the string matches one of the formats, the function should return the matching hours and minutes as "HH:MM". For example, if the string contains "15:42:38", we should return "15:42". If the string contains "2:30 PM", we should return "2:30".

If you are feeling ambitious, for the strings that match the second format, convert the result to 24-hour time before returning it. (So for "2:30 PM" we would return "14:30".)

If you're bored...

`http://software-carpentry.org/4_0/regexp/exercises/`

(Not required)