Plot review and identifying patterns for Algorithms: CS Assignment 10 and 11 review + prep

C. Woodford

January 25, 2018

1 Announcements

As most of the class struggled with Assignment 10 and it seems that your courseloads this term are becoming overwhelming, the structure for assignments for the foreseeable future will be for assignments given on Tuesdays and due the next Sunday, giving a 12 day span and 3 classes for you to talk with each other and to ask questions in class.

This is effective immediately. Assignment 10 has an extended deadline of Sunday Jan 28th at midnight, and Assignment 11 has a new deadline of Feb 4th. Your next assignment (Assignment 12) will be assigned on Tuesday Feb 6th.

Where everyone has already submitted something for Assignment 10, I have returned those scripts with grades and comments on how to improve. I recommend improving your scripts and resubmitting - the better of the two marks will be the kept grade.

Another important thing to keep in mind is that I cannot grade files I cannot open or convert. Please make sure that you send me a .py or .ipynb file for assignment submissions.

Lastly, be careful with complex assignments. It's better to tackle the assignment piece by piece and submit a working albeit unfinished script than a nonworking script with all pieces partially completed.

2 Plotting

As seen in the Assignment 10 description, there were 2 main parts to complete:

1. Plot the projectile in a meaningful manner
2. Save a plot for every 5 time steps

The first of these is what we'll cover here. It seems that the idea of plotting overall wasn't difficult, but deciding what to plot and how to plot it, as well as saving the figures instead of showing them.

The commands in particular to remember here are:

```
import matplotlib.pyplot as plt
plt.plot(x,y) or plt.scatter(x,y)
plt.xlabel()
plt.ylabel()
plt.title()
plt.savefig()
```

You need to think about what `x,y` should be for this case. Should they be arrays of all data points of the projectile? Should they be singular points at the time step you're plotting? You can experiment and see what is best.

The other thing to consider is what the name of the plots should be for `plt.savefig()`. You don't want to save to the same file every time as your plot will be continuously overwritten, so you'll need to find a way to update the name. This leads us to singling out the specific time step in the following section.

3 Identifying Algorithms

3.1 Singling out specific iterations

Singling out iterations is an important tool to have. Most if not all major codes will run on a series of loops, or at the very least have iterations at some point. Typically, the iteration step (in this example, the time step) is very small, and your loops may have many iterations. If you want output (either written or plots), you probably don't want it for every time step as it will get cumbersome, and fast. So you need to find a way to single out every 5 iterations, or 20, or 100.

Note that singling out iterations does NOT mean changing the iteration step. So in particular for Assignment 10, you should not need to change anything in the core file - only add lines for plotting and comments.

Thinking about how to do this, you need to find a pattern in the iterations. We discussed this at length in class and I have a follow-up section below with more details.

The obvious thing is we need to have an `if` statement inside the while loop to find when we've reached the correct iteration numbers. The question then is WHAT needs to be in our `if` statement as the condition to find these? Our basic structure is:

```
if x == ?  :
   # Plot and stuff
```

So what needs to fill in the `x` and `?` markers? Let's focus on `x`. This variable is what we need to compare to actually find the iterations we need, and so should have iteration information in it somewhere. The obvious choices are the variables already inside the while loop: `x,y,vx,vy,t`. We also know that our iteration step in this case is the time step `dt`, so this leads us to consider `t`, which is updated in every iteration using `t += dt`.

How do we extract iteration information from `t`? It may help to think about how `t` changes for each iteration and then look for a pattern. Thinking about the iteration number, we can see:

$$
\begin{aligned}
&Step\ 0 : t(0) = t_i \\
&Step\ 1 : t(1) = t(0) + dt = t_i + dt \\
&Step\ 2 : t(2) = t(1) + dt = t_i + 2dt \\
&Step\ 3 : t(3) = t(2) + dt = t_i + 3dt \\
&\qquad\qquad \vdots \\
&Step\ n : t(n) = t(n-1) + dt = t_i + ndt
\end{aligned}
$$

Can knowing the iteration number help us single out iterations? Let's take an example. Say we want to highlight every second iteration starting from the first iteration, to print or plot or do a calculation or what have you. What would that mean about the iteration number for each of the iterations we want to highlight? If we start counting at 0, then we would want to highlight iterations 0,2,4,6,8,10, ... An obvious pattern is that these numbers are even, or rather they are divisible by 2. If we're able to extract the iteration number and test whether it's divisible by 2 or not, we can determine whether or not to highlight that specific iteration. This would be:

```
if n%2==0:
   # Plot and stuff
```

where `n` is the iteration number. We don't typically have the iteration number lying around, but there are ways to extract it. Using `t`, you can reorganize the last line of the above expression for `n`, or you could initialize a counter to count the iteration steps as you go. Note that you'll need to have some way of updating the plot names, so a counter could be used for both.

3.2 Mathmatical Induction and Pattern identification

What we did in the previous section was a classic case of pattern recognition/identification. Remember the 5 steps for coding:

1. Identify the problem
2. Find a solution
3. Implement the solution
4. Improve efficiency
5. Improve aesthetic

Steps 1-3 are always required, and to identify a solution you typically need to identify a pattern in your problem. A good starting point is to implement what we did here: go through the first few iterations of a problem to see if there's a pattern that emerges. This may take some time, as there could be multiple things to check, but you should always start with the building blocks. In the case for Assignment 10, `t` was the simplest variable to choose, but we could have found a similar pattern in the other updating variables - they just would have been more difficult to use and decipher.

A nice tool for testing patterns is mathematical induction, although keep in mind it may not always be possible to implement it. You need to prove that the pattern works for the base case, typically where the step or first input is 1. You then need to prove that it is true for a general input n. The final test is that if the pattern works for the case 1 and the case n, it should also work for the case n+1. If it is true for all 3 (typically the case n will be a given or obvious), then the pattern should work for all inputs.

For example, with our case of `t`, we can make a statement that `t` at any given time step n will be `t(n) = ti + n*dt`. For n=1, we know that (depending on how you start the counting) `t(0) = ti` and `t(1) = t(0)+1*dt = ti + dt`. Then we consider the n+1 case: `t(n+1) = t(n) + dt = ti + n*dt+dt = ti + (n+1)*dt`, and so our pattern passes under mathematical induction.