Experimental Physics 1 - Python Problem Set 2

October 20, 2018

1 Tanks for the memories!

One of the thrills of being old is that you can remember when people played really awesome games on computers instead of Fortnite or whatever it is you kids do for fun these days. Like, when I was in school, they tried to make you play Oregon trail, because it was educational (spoiler: everyone died of dysentery). But an awesome game you could play instead was Artillery, where you took turns trying to shoot each other with tank shells. Physics at work!

So here's the way artillery works. You have a tank. Your opponent has a tank. You input an angle and a velocity, which launches a shell at your opponent. If you hit your opponent's tank, you win. If not, your opponent gets a turn. This goes on until someone wins, or the teacher makes you play Oregon Trail. To make it a little more interesting, there was usually a wall of some kind in between the tanks.

We're going to write this game. It seems really complex, but it's actually a matter of just breaking it down into small parts.

1.1 The Physics

If you fire a shell at an initial velocity $v(0)$ an angle θ from the horizontal, you can break the initial velocity down into its x and y components

$$v_x(0) = v(0)\cos(\theta) \tag{1}$$
$$v_y(0) = v(0)\sin(\theta) \tag{2}$$

If the shell originates from x_0, y_0, the position vs. time will be given by the formulae

$$x(t) = x_0 + v_x(0)t \tag{3}$$
$$y(t) = y_0 + v_y(0)t - \frac{1}{2}gt^2 \tag{4}$$

Let's say that our world is bounded below at y = 0, so as soon as the shell hits y = 0, we can stop worrying about its path. We can find this time by

$$y(t_{final}) = y_0 + v_y(0)t_{final} - \frac{1}{2}gt_{final}^2 \tag{5}$$
$$0 = \frac{1}{2}gt_{final}^2 - v_y(0)t_{final} - y_0 \tag{6}$$
$$t_{final} = \frac{v_y(0)}{g} + \sqrt{(\frac{v_y(0)}{g})^2 - 2\frac{y_0}{g}} \tag{7}$$

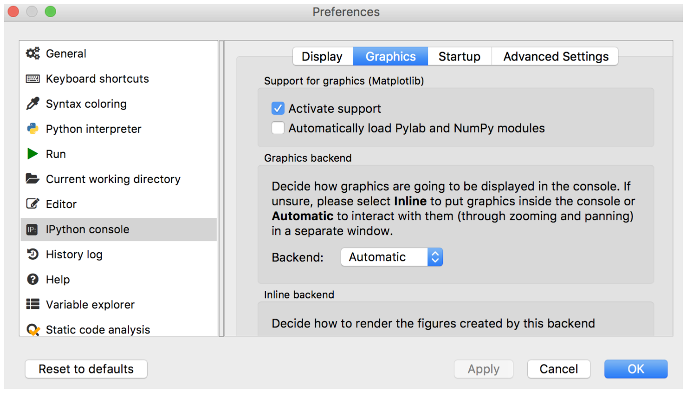
1.2 Some reminders from last week

Place

```
import numpy as np
import matlplotlib.pyplot as plt
```

at the beginning of your code.

By default, Spyder places plots inline in the console instead of in separate windows. To get your plots into separate windows, you have to change the settings. Choose **preferences** (under the python menu or the tools menu depending on your OS), click on the **IPython console** tab on the left, then the **Graphics** tab on the top. Look for **Graphics backend**, then choose **automatic**. Click on **OK**, then **quit and restart** spyder



Support for graphics (Matplotlib)

[✓] Activate support

[ ] Automatically load Pylab and NumPy modules

Graphics backend

Decide how graphics are going to be displayed in the console. If unsure, please select **Inline** to put graphics inside the console or **Automatic** to interact with them (through zooming and panning) in a separate window.

Backend: Automatic

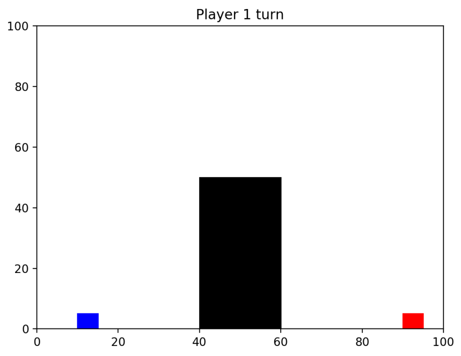
Inline backend

Decide how to render the figures created by this backend

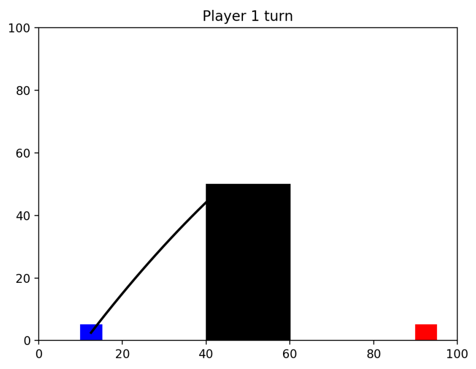
Places you can go for help:

- `https://matplotlib.org/` - matplotlib website, which has helpful info, including
  - `https://matplotlib.org/tutorials/index.html` - matplotlib tutorials, especially ...
  - intro to pyplot
  - `https://matplotlib.org/gallery/index.html` - a gallery of pyplot examples; very sophisticated but also inspiring?
- `http://www.physics.nyu.edu/pine/pymanual/html/chap5/chap5_plot.html` - the plotting chapter from the Pine manual
- In general, the Pine Manual is helpful, but it's written for Python 2. If something from the manual doesn't run, that's the likely culprit
- Your book discusses plotting. It uses "Pylab" which is just a wrapper for numpy and matplotlib together. Anywhere you see "Pylab," just make the appropriate substitution

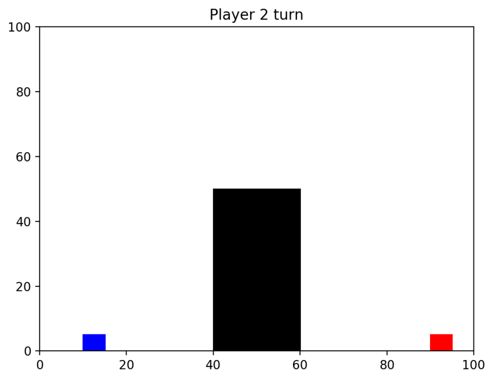
2 Example Game Play



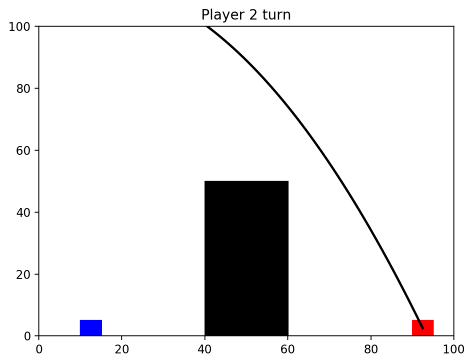
```
Player 1 enter velocity >50
Player 1 enter angle (deg) >60
```



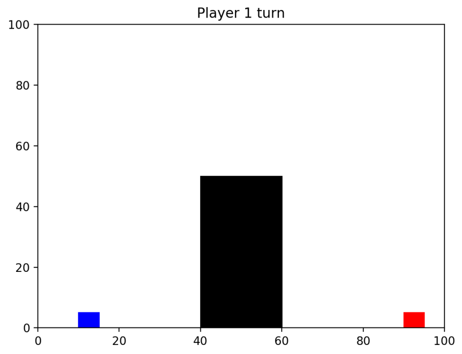
```
hit enter to continue >
```



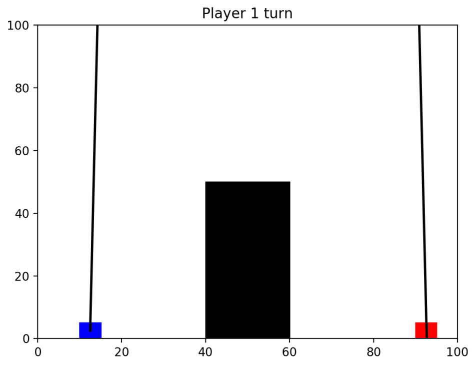
```
Player 2 enter velocity >50
Player 2 enter angle (deg) >110
```



```
hit enter to continue >
```



```
Player 1 enter velocity >150
Player 1 enter angle (deg) >89
```



```
Congratulations player 1 !
```

3 Top down design of the program

Let's break this problem down into progressively smaller chunks. This design strategy is called top down.

3.1 Highest Level - the game

1. The game needs to pick or be given initial locations for the tanks and obstacle.
2. The game needs to give one player a turn and detect if that player has won
3. If a player has won, the game needs to print a congratulatory message and end. Otherwise, the other player gets a turn (go back one step).

3.2 Next Highest Level - the turn

Steps 1 and 3 are pretty small tasks already, so we don't need to break them down further. Step 2 needs to be broken up into smaller chunks.

1. At the beginning of the turn, erase the figure window and draw the game board.
2. Ask the player whose turn it is for an angle and a velocity.
3. Calculate and draw the outcome of the shot.
4. Return to the caller whether there was a winner

3.3 Down one more level

Step 2 - asking for input, and step 4 returning our answer are already pretty small chunks, so let's look at the other two

3.3.1 Drawing the board

1. Draw tank 1
2. Draw tank 2
3. Draw the obstacle

We could be quite elaborate, but we'll just draw each tank and the obstacle as a filled rectangle of a different color, ***and I'll write the function for you***

3.3.2 Calculate and draw the outcome of the shot

1. Calculate a trajectory from the center of the tank using equations of subsection 1.1.
2. Calculate if the trajectory hits the obstacle. If it does, shorten the trajectory to end at the obstacle.
3. Draw the shortened trajectory on the screen.
4. Calculate if the shortened trajectory hits the opposing tank. If it does, return victory code. If it does not, return failure code.

3.4 At the very lowest level

You will need to write a function to calculate the trajectory that stands alone.

You will need to write a function that finds the first point on the trajectory to intersect a box. You can use this to see if the shot hits the obstacle and to see if it hits the opposing player's tank.

4 How to represent the game

We will need to figure out how to represent several elements of the game as data.

- The tanks – the tanks have a location. The shells need to originate from the firing tank, and we need to test whether the shell has hit the opposing tank. Let's represent the location of each tank as a box, which we'll put into data as a *tuple* of four elements: (left,bottom,right,top) or (x0, y0, x1, y1), where x0<x1 and y0<y1.
- The obstacle can also be a box, represented the same way, as a tuple of (left,bottom,right,top).
- We need to represent the trajectory of the fired shell. We'll use a np.array for the x coordinates and a separate np.array for the y coordinates. Each index will represent a step forward in time. So $(x[j], y[j])$ is a point on the trajectory, and the shell travels from $(x[0], y[0])$ to(x[N−1],y[N−1]), where N is the number of trajectory points.
- The player whose turn it is will be an int, either 1 or 2.
- The acceleration due to gravity is a float g, with a default value of 9.8. The velocity and angle of the shell are floats. The user will enter the angle in degrees but you need to do calculations in radians (hint: np.deg2rad)

5 Turn the top down decomposition into functions

Now let's take our top down design and turn it into functions. You will find a function signature (the part that starts def) and docstring for each function below in the starter code. Your job is to write the implementation that fills in the details

Each numbered or lettered line below should correspond to one or two lines of code in your program. That's the beauty of breaking things down into small chunks. If it seems to you that any one line is very complex, get help with that line!

5.1 Highest Level - the game

```
def playGame(tank1box, tank2box, obstacleBox, g = 9.8):
    """
    parameters
    ----------
    tank1box : tuple
        (left,right,bottom,top) location of player1's tank
    tank2box : tuple
        (left,right,bottom,top) location of player1's tank
    obstacleBox : tuple
        (left,right,bottom,top) location of the central obstacle
    playerNum : int
        1 or 2 -- who's turn it is to shoot
     g : float
        accel due to gravity (default 9.8)
    """
```

What your implementation needs to do:

1. Locations of the tanks and obstacle are given as arguments to the function.
2. set playerNum to 1
3. loop forever:
   (a) do one turn for player playerNum (oneTurn) and capture the return value
   (b) if the player won break
   (c) put in a prompt to hit enter to continue – this keeps the screen from getting cleared too fast on next turn
   (d) change playerNum to opposite number (1⟶, 2⟶1)
4. Congratulate playerNum

5.2 Next Highest Level - the turn

```
def oneTurn (tank1box, tank2box, obstacleBox, playerNum, g = 9.8):
    """
    parameters
    ----------
    tank1box : tuple
        (left,right,bottom,top) location of player1's tank
    tank2box : tuple
        (left,right,bottom,top) location of player1's tank
    obstacleBox : tuple
```

```
        (left,right,bottom,top) location of the central obstacle
    playerNum : int
        1 or 2 -- who's turn it is to shoot
     g : float
        accel due to gravity (default 9.8)
    returns
    -------
    int
        code 0 = miss, 1 or 2 -- that player won

    clears figure
    draws tanks and obstacles as boxes
    prompts player for velocity and angle
    displays trajectory (shot originates from center of tank)
    returns 0 for miss, 1 or 2 for victory
    """
```

1. Erase the figure window and draw the game board. (drawBoard)
2. Ask the player whose turn it is for an angle and a velocity. Use getNumberInput, which is provided in the starter code.
3. if playerNum is 1, then origin = tank1box and target = tank2box
4. otherwise, origin = tank2box and target = tank1box
5. x0,y0 are the center of the origin box
6. Calculate and draw the outcome of the shot. (tankShot)
7. If there was a hit, return playerNum indicating a winner, otherwise return 0.

5.3 Down one more level

5.3.1 Drawing the board

```
def drawBoard (tank1box, tank2box, obstacleBox, playerNum):
    """
    draws the game board, pre-shot
    parameters
    ----------
    tank1box : tuple
        (left,right,bottom,top) location of player1's tank
    tank2box : tuple
        (left,right,bottom,top) location of player1's tank
    obstacleBox : tuple
        (left,right,bottom,top) location of the central obstacle
    playerNum : int
        1 or 2 -- who's turn it is to shoot

    """
```

1. clear the figure window (plt.clf())
2. Draw tank 1 (drawBox is provided in starter code)
3. Draw tank 2 (drawBox)

4. Draw the obstacle (drawBox)
5. Set the axes to display the range 0 to 100 in x and y. (plt.xlim , plt.ylim)

5.3.2 Calculate and draw the outcome of the shot

```
def tankShot (targetBox, obstacleBox, x0, y0, v, theta, g = 9.8):
    """
    executes one tank shot

    parameters
    ----------
    targetBox : tuple
        (left,right,bottom,top) location of the target
    obstacleBox : tuple
        (left,right,bottom,top) location of the central obstacle
    x0,y0 :floats
        origin of the shot
    v : float
        velocity of the shot
    theta : float
        angle of the shot
    g : float
        accel due to gravity (default 9.8)
    returns
    --------
    int
        code: 0 = miss, 1 = hit

    hit if trajectory intersects target box before intersecting
    obstacle box
    draws the truncated trajectory in current plot window
    """
```

1. Calculate a trajectory from x0,y0 at the given angle and velocity (x,y = trajectory (....))
2. Calculate if the trajectory hits the obstacle. If it does, shorten the trajectory to end at the obstacle.(x,y = endTrajectory.
3. Draw the shortened trajectory on the screen. (plt.plot)
4. return whether firstInBox(x,y,targetBox) is greater than or equal to 0 (a hit!)

5.4 At the very lowest level

5.4.1 Trajectory

```
def trajectory (x0,y0,v,theta,g = 9.8, npts = 1000):
    """
    finds the x-y trajectory of a projectile

    parameters
    ----------
    x0 : float
        initial x - position
    y0 : float
```

```
        initial y - position , must be >0
        initial velocity
    theta : float
        initial angle (in degrees)
    g : float (default 9.8)
        acceleration due to gravity
    npts : int
        number of points in the sample

    returns
    ---------
    (x,y) : tuple of np.array of floats
        trajectory of the projectile vs time
    """
```

1. Calculate $vx = v\cos(\theta)$
2. Calculate $vy = v\sin(\theta)$
3. Calculate $t_{final} = \frac{v_y(0)}{g} + \sqrt{(\frac{v_y(0)}{g})^2 - 2\frac{y_0}{g}}$
4. create an evenly spaced time axis with npts points between 0 and t_{final} (hint: t = np.linspace (...))
5. $x = x_0 + vxt$
6. $y = y_0 + vyt - \frac{1}{2}gt^2$
7. **return** (x,y)

5.4.2 FirstInBox

You will need to write a function that finds the first point on the trajectory to intersect a box. You can use this to see if the shot hits the obstacle and to see if it hits the opposing player's tank.

```
def firstInBox (x,y,box):
    """
    finds first index of x,y inside box

    paramaters
    -------------
    x,y : np array type
        positions to check
    box : tuple
        (left ,right ,bottom ,top)

    returns
    ---------
    int
        the lowest j such that
        x[j] is in [left ,right] and
        y[j] is in [bottom,top]
        -1 if the line x,y does not go through the box
    """
```

1. use a for loop to send j from 0 to the length of x

    (a) if x[j] is between box[0] and box[1] and y[j] is between box[2] and box[3] return j

2. if you make it to the end of the for loop without returning, that means there is no set of x,y in the box. return -1

I have given you a function in the starter that uses firstInBox to shorten the trajectory to the first point of intersection with a box.

```
def endTrajectoryAtIntersection (x,y,box):
    """
    portion of trajectory prior to first intersection with box

    paramaters
    ----------
    x,y : np array type
        position to check
    box : tuple
        (left ,right ,bottom ,top)

    returns
    ----------
    (x,y) : tuple of np.array of floats
        equal to inputs if (x,y) does not intersect box
        otherwise returns the initial portion of the trajectory
        up until the point of intersection with the box
    """
    i = firstInBox(x,y,box)
    if (i < 0):
        return (x,y)
    return (x[0:i],y[0:i])
```

6 Bottom Up Implementation

Now that you know everything you need to do, it's time to do it. Although it's great to design fromt the top-down, it's better to implement from the bottom up. That way, as you build each component, you can test it individually, using only the parts you've already finished.

1. Write the trajectory function. Test it by plotting some (x,y) curves. Can you start at arbitrary x and y points (as long as y > 0?). Does the answer make sense at all angles between 0 and 360?

2. Write firstInBox (endTrajectoryAtIntersection is written for you). Test it by drawing a trajectory, picking a box on the trajectory, and then seeing if the trajectory ends at the right spot.

3. Write drawBoard. Test it by drawing the board.

4. Write tankShot. Pick one example shot that starts at (1,1) and ends somewhere on the board (between x = 0 and x = 100). Test it with various locations for the obstacle and the opposing tank. Make sure it returns true if you hit the opposing tank and false otherwise.

5. Write oneTurn. Test it with some of your example tank locations and shots. Note that if tank1 is to the left of tank2, then tank1 will generally fire between 0 and 89 degrees and tank2 will fire between 91 and 180.

6. Write playGame. Test it by playing against yourself or your friend (take away their Fortnite). Make sure the game ends on a hit and the victory message displays.

I've included in the starter code tester functions for the trajectory and for when you've finished tankShot and drawBoard. But you should feel free to write your own testers too!

7 Extensions

Some ideas to make your tank game more fun.

- Randomly initialize the board with different locations and obstacles
- Add wind (add to v_x) that changes from turn to turn with an arrow to indicate strength and direction
- Multiple obstacles
- A player can lose the game if (s)he shoots his/her own tank
- Better graphics
- Animate the shots
- Make a computer opponent.