Phys 265 - Lab 1A – Electrostatic Projectile Game

Download and edit lab1a_lastname.py, which is posted on ELMS. Submit three items to your github account:

1. *Your completed version of lab1a_astname.py (re-named appropriately)*
2. *A screenshot of your desktop after running your code.*
3. *A two-page summary of your final submission. See below for more information.*

Introduction:

You are prototyping a game intended to be played by physics and astronomy students. The goal of the game is to deduce the location of several electric charges that are hidden in the xy plane. The player will launch charged projectiles, and by observing the trajectories, he or she will try to deduce the location of the hidden charges.

Your task is to solve Newton's 2nd Law using `solve_ivp()` to compute the trajectories of projectiles subject to the electrostatic force. You will be using pyplot to plot the potential function and the trajectories.

Once you have completed your game prototype, you will submit the python code along with a two to three page prospectus to your boss at your video game company. She will decide whether the game should be fully developed and marketed. More information about the prospectus you will write can be found at the end of this document.

Physics Background:

A charged projectile is experiencing an electrostatic force that decreases as 1/r:

$$\vec{F}(r) = \frac{kQ}{r}\hat{r}_{12}$$

where Q is the magnitude of the hidden charge which is creating the force, k is the coupling constant, $\hat{r}_{12}$ is a unit vector pointing from the location of charge Q to the field point where the projectile is located, and r is the distance from the charge to the projectile. Q may be positive or negative, so the force may be attractive or repulsive. If multiple charges are present, then the total force is the vector sum of the individual forces.

The potential function for this force law is

$$\varphi(r) = kQln(r_0/r)$$

where r_0 is a reference location where the potential is set to zero. We will choose $r_0 = 1$, and k = 1000. When multiple charges are present, the total potential is the scalar sum of the potentials due to each charge. The force law given above is the gradient of this potential function.

Code Files

Download from ELMS lab1a_utilities.py and lab1a_lastname.py

lab1a_utilities.py contains two functions: `calculate_force(x,y,hidden_charges)` and `calculate_potential(x,y,hidden_charges).` You do not need to edit these functions. They calculate the force and potential as indicated in the two physics equations above.

The file lab1a_lastname.py contains the skeleton code that you will edit. Download this code from ELMS and rename it appropriately.

Part 1 – Contour Plot of the Potential Function

In this section you will place two charges in the xy plane and create a contour plot of the potential function.

1) While developing your game prototype, you will not be using the green arrow 'run' buttons at the top of the spyder window. Instead, you will import your code in the ipython console and run the functions contained in your code from the ipython console.

2) To read the instructions (docstrings) for how to use the two functions contained in lab1a_utilities.py, do the following at the ipython console:

```
> import lab1a_utilities as util
> util.calculate_potential?
```

   This will print out to console the docstring for this function. Take note of the input parameter types and descriptions, and also the description of what the function returns.

3) At the ipython console, do the following:

```
> import lab1a_lastname as game
```

   You should see an empty game window appear on your desktop with a gridded xy plane.

4) In the editor, open lab1a_lastname.py. At the top of the file, create a 2D numpy array to specify two hidden charges. Refer to the docstring for `calculate_potential()` for more information on how to specify this 2D array. Start with charges that have magnitude +1 or -1; you can change these later. You may choose their xy locations as you wish.

5) Save lab1a_lastname.py in the editor, and import it again at the ipython console:

```
> import lab1a_lastname as game
```

6) Now you will write a function that will plot the potential of your two charges in the Game Window. Add statements to the skeleton code of `reveal_potential()`, located in lab1a_lastname.py, to make a contour plot of the potential function. Inside

`reveal_potential`(), you should create an xy domain with `np.meshgrid(),` loop manually over the xy domain, and call `calculate_potential()` to get the numerical values of the potential function. Store the potential function in a 2D array and make a contour plot of it in the game window.

7) Test your code for `reveal_potential()` by saving your code in the spyder editor, then doing this at the ipython console:

```
> import lab1a_lastname as game
> game.reveal_potential()
```

This should draw your contour plot of the potential in the Game Window.

8) Recall the `'levels'` parameter of `ax.contour().` This parameter allows us to define a set of equipotential curves in the contour plot. The grammar is `levels=np.linspace(lowerlimit,upperlimit,number).` Set and adjust the levels parameter so that you see a reasonable number of equipotential curves on your potential plot.
    a. Remember, each time you edit lab1a_lastname.py, you need to save it, then repeat the following statements in the ipython console in order to run the updated code:

```
> import lab1a_lastname as game
> game.reveal_potential()
```

    b. Hint: To get a reasonable number of equipotentials to show up on your plot, it is often helpful to see a 3D plot of the function before setting the levels parameter. You can temporarily make a 3D wireframe or surface plot:
        i. At the top of `reveal_potential(),` comment out the 'ax = fig.axes[0]' statement.
        ii. Uncomment the two other statements: 'fig.clf()', and 'fig.add_subplot(projection='3d').
        iii. This should allow you to add a statement to `reveal_potential()` to make a 3D wireframe plot or a surface plot of the potential function in the Game Window. When the plot is made, you can make a note of the function maximum and minimum to help you choose the levels parameter appropriately.
        iv. Each time you want to run your code, be sure to save lab1a_lastname.py and then re-import it in the ipython console.
        v. When you are done, return to the 2D contour plot. Undo the commenting and uncommenting indicated above.

9) Optional: Add code to `reveal_forcefield()` to make a `streamplot` of the calculate_force() function.

Part 2 – Trajectory Calculation

In this section you will add code to `plot_trajectory()` to compute the motion of the projectile using `solve_ivp()`. The initial conditions for this 2-dimensional 2nd order problem are `x0, y0, vx0`, and `vy0`. However, instead of specifying `vx0` and `vy0` directly, we will specify the magnitude of the velocity vector (`v0`) and the angle that it makes with the x-axis (`angle`). The default initial conditions are given in lab1a_lastname.py.

1) Inside `plot_trajectory()`, write a `derivatives()` function to solve Newton's 2nd law for the projectile. To get the x and y components of the force, call the `calculate_force()` function, which is given to you in lab1a_utilities.py.

2) Add code to `plot_trajectory()` to compute `vx0` and `vy0` from `v0` and `angle`. Note: `angle` should be stored in units of degrees, not radians.

3) Define a start time and a stop time.

4) Call `solve_ivp()` to compute the projectile trajectory. Plot the trajectory as a line on the figure. (This should be done inside `plot_trajectory()`.)

5) Make sure that `plot_trajectory()` ends with a `fig.show()` statement. Then try exercising your code by calling `plot_trajectory()` from the ipython console:

```
> import lab1a_lastname as game
> game.plot_trajectory()
```

6) Also call `reveal_potential()` from the console so you can visualize the force that the projectile is experiencing. Inspect the trajectory to see if it makes sense. Adjust the stop time if you want to see more or less of the trajectory.

7) If necessary, adjust the magnitude and the location of the charge so that it has a noticeable effect on the projectile. You may also need to adjust the `levels` parameter of `plt.contour()` so that the potential function remains visible.

8) Add black dots to your plotted trajectory, evenly spaced in time. Choose a reasonable time interval so that the dots are not too dense nor too sparse. Recall that black dots can be added with array slicing and by using the `'ok'` option of `ax.plot()`. You can adjust the size of the dots with the `markersize` parameter of `ax.plot()`.

9) The function `play()` to is given to you in lab1LastName.py. It allows the user to adjust the initial conditions on `v0`, `angle`, and `y0`. Call `play()` several times from the console and observe the different trajectories that you get with different initial conditions. If you want to erase the trajectories, call `clear()`, which is also given to you in lab1lastname.py.

Part 3 – Game Solution

In this part you will adjust the charges to optimize the game, and write a function to allow the user to guess where the charges are located in the xy plane.

1) First add two to five additional charges to the xy plane by modifying your charges array in lab1a_lastname.py. The final number of charges should be between three and six.

2) Try playing the game several times using a variety of initial conditions. You should aim to design the game to be neither too easy nor too hard. Adjust the location of the charges and their strength so that the player has a reasonable chance to find them, but without making the game trivial. Keep in mind that charges that are far away from x0 = -100 (the starting location) may be difficult for the player to find. Also, it's probably a good idea to locate all the charges between -100 and +100 in both x and y, so that they are not too close to the boundary.

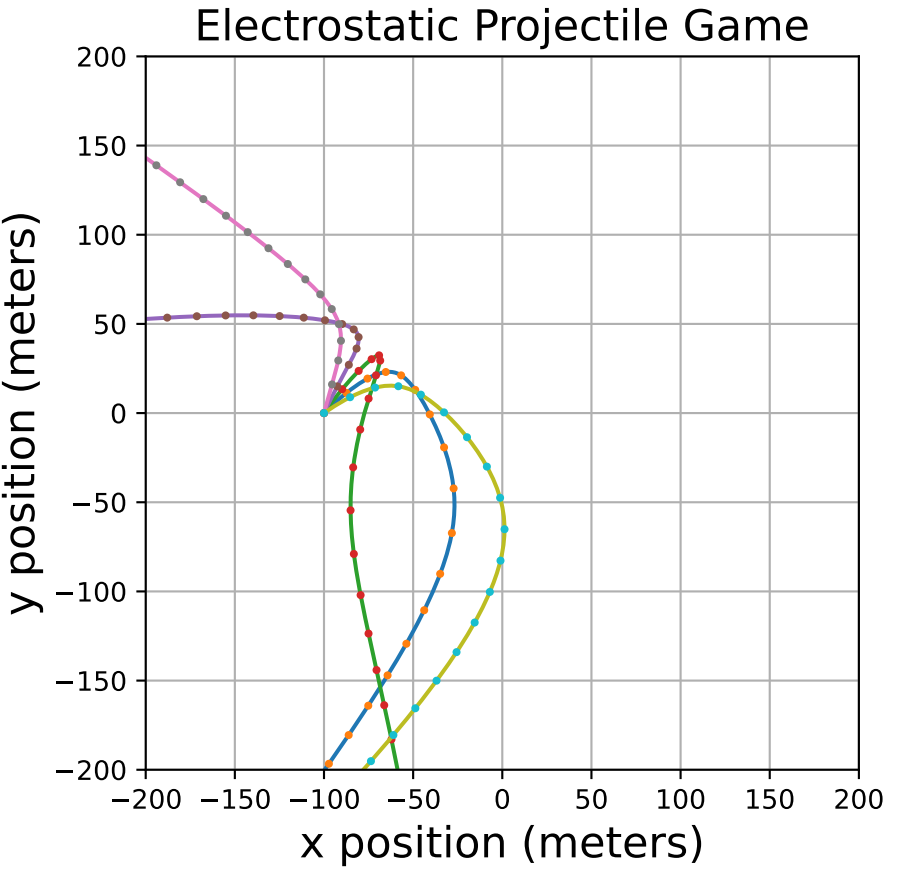
3) Add code to `solve_it()` to allow the user to guess where the charges are located. Do this by asking the user four questions, one question for each quadrant of the plane: Does quadrant 1 have (1) positive charge; (2) negative charge; (3) neither; or (4) both? How about quadrant 2? Ect. The user should answer each question by entering an integer between 1 and 4 as indicated. For each quadrant, compare the user's answer to the true locations of the charge, and tell the user if he/she is correct or incorrect.

4) If all four answers are correct, print out a message of congratulations, and call `reveal_potential()` to plot the potential. If any of the answers are incorrect, print out 'Sorry, try again.'

Final submission: Submit your code on github along with your two to three page prospectus describing your game and a screen shot of your desktop. Your prospectus should explain the nature of the game, how you implemented it, and how the player can correctly deduce the locations of the charges. Include plots of sample trajectories that illustrate how the user should be able to deduce the locations of the charges.

Your boss, who will read the prospectus, has some knowledge of scientific programming and python, but she is not an expert on numpy, pyplot, and scipy. Your prospectus will need to give some basic explanations of how the code is working and how the game is played. Your prospectus should include the following sections:

(1) Introduction; (2) Game Physics; (3) Implementation (What each function does and how it does it); (4) Game Play (What the player will see and experience, including plots of the equipotentials and/or the force field, and one or more sample trajectories); and (5) Game Solution: (How the user can solve the game, including example trajectories, and how they allow the player to find the charges.)


Electrostatic Projectile Game
y position (meters)
x position (meters)
200
150
100
50
0
−50
−100
−150
−200
−200 −150 −100 −50 0 50 100 150 200