PHYS 3926: Project 2

Aidan Sirbu
asirbu@uwo.ca

Western University — September 22, 2021

1 Introduction

The purpose of this simulation is to model the projectile motion of a baseball being struck by a *Robotic Designated Hitter (RDH)*. The simulation has the option to dynamically account for air resistance depending on the ball's velocity at every point. To implement the set of ordinary differential equations given:

$$\frac{d\vec{r}}{dt} = \vec{v} \tag{1}$$
$$\frac{d\vec{v}}{dt} = -g\hat{y} - \vec{v}\frac{C_d \rho A v}{2m} \tag{2}$$

one must first use forward derivatives in order to arrange the ODE's into a form that suits numerical solvers such as Euler's method. Then, the *x* and *y* components were split into separate equations so that a computer may deal with them separately:

$$r_x(t+\tau) = r_x(t) + \tau v_x \tag{3}$$
$$r_y(t+\tau) = r_y(t) + \tau v_y \tag{4}$$
$$v_x(t+\tau) = v_x(t)\left(1 - \tau\frac{C_d \rho A\sqrt{v_x^2 + v_y^2}}{2m}\right)\hat{x} \tag{5}$$
$$v_y(t+\tau) = v_y(t)\left(1 - \tau\frac{C_d \rho A\sqrt{v_x^2 + v_y^2}}{2m}\right)\hat{y} - \tau g\hat{y} \tag{6}$$
$$\tag{7}$$

Also, note that throughout the simulation, α is defined as:

$$\alpha = \frac{C_d \rho A\sqrt{v_x^2 + v_y^2}}{2m} \tag{8}$$

For the purposes of this simulation, three functions were created. `projectile` plots the projectile motion of the baseball and returns a horizontal range in meters. `projectileFence` plots the projectile motion of the baseball but returns the height of the ball at 400 feet so that the user may determine whether or not the baseball flies over the back fence. Finally, `abhr` determines the at-bat, home-run ratio depending on a variety of parameters. To achieve this, 3 methods of numerically solving ODE's were implemented, Euler, Euler-Cromer, and Midpoint methods. An object-oriented approach to this problem was taken. This means that every requirement of this simulation is handled by a set of 3 functions. Each function has an associated comment block specifying every parameter and what its output will be. The point of this approach is to ensure re-usability of this code in any scripts a user may be designing. The user may yield various plots, ranges, and at-bat home-run ratios simply by making various function calls.

2 Tasks

2.1 Part 1

To accomplish task 1, the function `projectile` was created. The input parameters are: initial velocity, initial launch angle, time step, air (which allows the user to choose whether or not air resistance should be accounted for), method (which allows the user to specify which numerical solver method should be used), and plot (which allows the user to specify whether or not a plot should be returned). Firstly, the function defines constants needed as well as initial conditions which are partly supplied by the user and partly a characteristic of the RDH (such as initial position). Next, the function determines the first α instance if the user chose to account for air resistance. Next, 3 loops were created, each dealing with a different method for solving the ODE's. Firstly where is the Euler loop:

```
while redundant == 1:
    # Position step
    rx = rx+tau*vx
    ry = ry+tau*vy
    # Velocity step
    vx = vx*(1-tau*alpha)
    vy = vy*(1-tau*alpha)-tau*G
    v = [vx, vy] # Update velocity vector
    # Termination criterion
    if ry <= 0:
        break
    r = [rx, ry] # Update position vector
    velList.append(v) # Update memory
    posList.append(r) # Update memory
```

As one can observe, Euler's loop first finds the next position vector using the previous position and velocity vectors. Next, it obtains the next velocity vector using the previous velocity vector. The loops are set to run until the break statement is reached. If the y-position of the ball is negative, the entire iteration is discarded and not appended to the velocity or position lists which will be used to plot the results later. The only variation between the Euler loop and Euler-Cromer loop is that the new velocity vector is found first, then the new position vector is found using the old position vector as well as the new, updated velocity vector. This can be observed in the listing below:

```
while redundant == 1:
    # Velocity step
    vx = vx*(1-tau*alpha)
    vy = vy*(1-tau*alpha)-tau*G
    v = [vx, vy] # Update velocity vector
    # Position step
    rx = rx+tau*vx
    ry = ry+tau*vy
```

Finally, the Midpoint loop first calculates the new velocity vector such as the Euler-Cromer loop. However, when determining the next position vector, it uses the old position vector and *an average* between the previous and updated velocity vector.

The termination criteria for the ODE solver loops was chosen to be:

```
if ry <= 0:
    break
```

This means that if the y-component of the position vector falls below 0, then the loop breaks before appending the next position into the results (which would be erroneously negative). However, this yields a new problem concerning the proper range of the ball as the last position vector will most likely be above $y = 0$. In order to approximate the correct range, the last trajectory of the ball – being the path from the last position vector to $y = 0$ – is assumed to follow a linear path. In order to do this, the function takes the last two position vectors and uses them to fit a line of best fit using a linear equation. The goal is the find what is *x* at *y* = *0*. The following formula is derived as:

$$y - y_1 = m(x - x_1) \tag{9}$$

$$y - y_1 = mx - mx_1 \tag{10}$$

$$x = x_1 - \frac{y_1}{m} \tag{11}$$

Therefore, all that is needed is the slope between the last two positions *m*, as well as the last position vector (x_1, y_1). To achieve this the following code was implemented:

```
if method != "all": # If all methods are used, multiple ranges will be reached, only a plot is
    needed
   lastIndex = len(posList) - 1
   xi = posList[lastIndex - 1]
   xf = posList[lastIndex]
   slope = (xf[1]-xi[1])/(xf[0]-xi[0])
   distance = xf[0]-xf[1]/slope
```

To demonstrate the effectiveness of the `projectile` function, figure 1 was created using all 3 ODE numerical methods with air resistance accounted for.

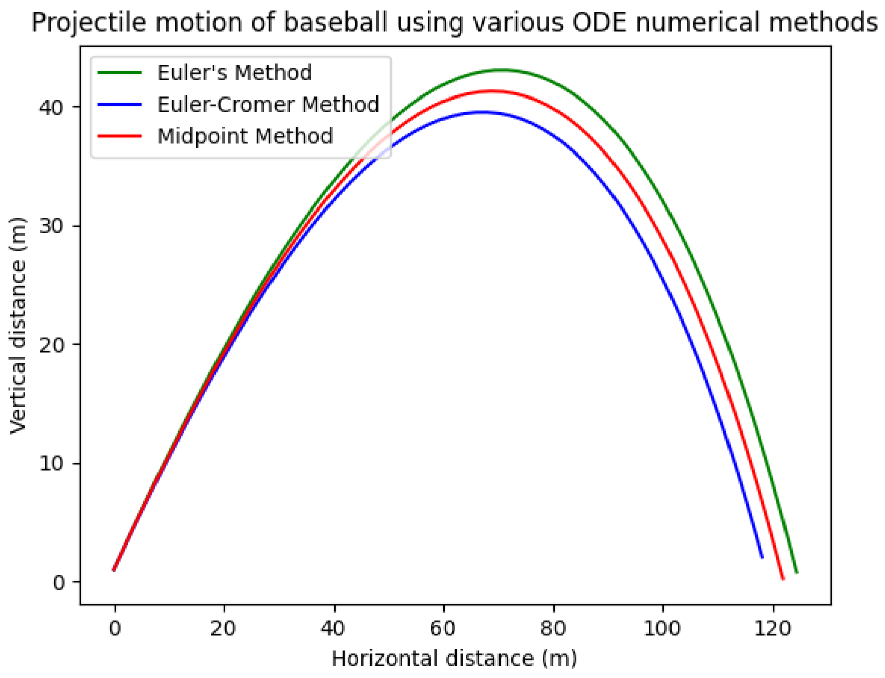

Figure 1: Represents the projectile motion of a baseball with initial velocity of 50m/s, launch angle of 45°, time step of 0.1s, and air resistance accounted for using the ODE numerical solving methods listed in the legend.

2.2 Part 2

The next objective of this investigation was determining the AB/HR-ratio for the proposed RDH. AB/HR-ratio is the at-bat, home-run ratio. This is a ratio between the amount of times the RDH is hitting the ball and the amount of times it leads to home-runs. The smaller the ratio, the better the player is. To determine this, the initial velocity and launch angle are pulled from random distributions with means of 100mph and 45° as well as standard deviations of 15mph and 10°. Function `abhr` was developed in order to calculate this. The intake parameters are the number of at-bats – the number of iterations of hits the function should calculate – whether air resistance should be accounted for (default being that it is accounted for), the method to be used (default Euler's method) as well as whether or not a back fence should be account for (default no fence), more on this in section 2.3. The determining factor for a home-run is if the horizontal range of the ball exceeds 400 feet, or in the case of the presence of a fence of given height, if the ball flies over the fence. Function `abhr` utilizes a `for` loop with a range of the at-bats to determine the desired ratio. Furthermore, there is also exception handling in the case that no home-runs were achieved as this would otherwise produce a zero-division error common for low at-bat numbers. The time-step of the iterations are 0.1s. The AB/HR ratio for the RDH was determined to be 3.16 over 10,000 at-bats.

2.3 Part 3

The final part of this investigation requires the simulation to account for the possibility of a back fence placed at 400 feet. The requirement of a home-run with a fence accounted for is that the ball flies over the fence. To find the AB/HR of the RDH with a fence, function `projectile` was modified into a secondary function `projectileFence` which takes the same parameters, but returns the height of the ball at either 400 feet, or the height of the ball once it has fallen below the *y-axis*, whichever comes first. This function also does not have the option to use all 3 ODE solving methods. This function was implemented within function `abhr` such that when `abhr` is called with a fence height other than 0, it determines home-runs based on the above criteria. The investigation in this section lies in the determination of how fence height affects the AB/HR ratio as well as finding the height of the fence required in order for the RDH to have an AB/HR ratio greater than 10. The former was found by analyzing the AB/HR for various fence heights and the trend is shown in figure 2.

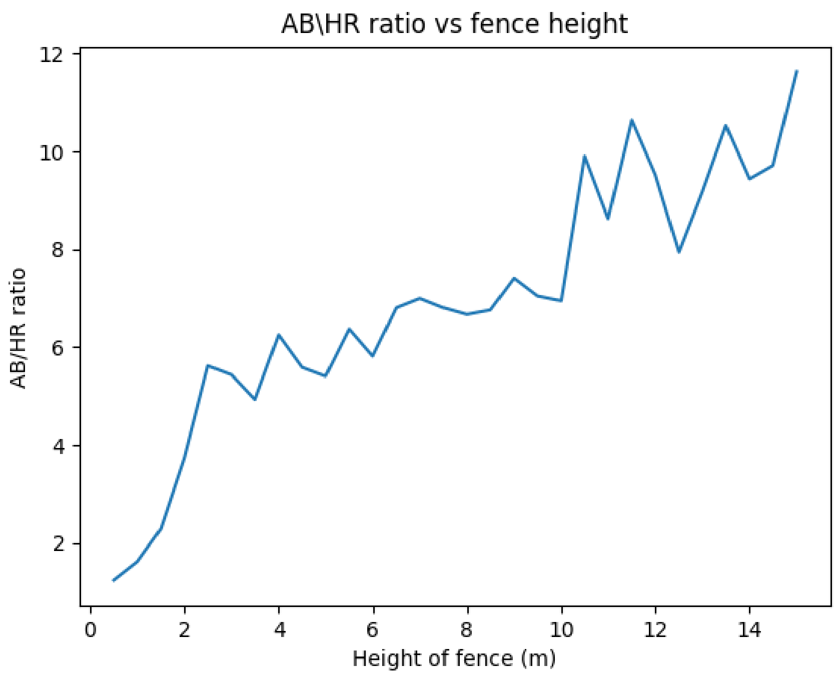

Figure 2: How AB/HR ratio varies depending on the fence height located at 400 feet.

As one can observe, the general trend is upwards as expected. As the fence height increases, so does the AB/HR ratio, meaning home-runs decrease. The trend however, it not smooth and very jagged. This is most likely due to the random nature of the home-runs as it is based off a random distribution of initial velocities and launch angles. Next, a script was written which repeatedly calls `abhr`, increasing the height of the fence by 1 at every iteration until the function returns a ratio greater than 10. At that point, it undergoes the process again for a total of 100 iterations in order to average the required fence height. Running this script yields an average required fence height of 14.0 meters to yield an AB/HR ratio greater than 10.

3 Summary of Results

To conclude, the effectiveness of the simulation has been shown. The program can find and plot the trajectory of a baseball hit by the RDH given any launch velocity and angle using Euler, Euler-Cromer, and Midpoint method for numerically solving ODE's. Figure 1 portrayed the variety in trajectory based on which method was chosen to determine it. While all methods yielded very similar trajectories at first, there was a divergence around 40 meters where each finds a different maximum height and final range. Furthermore, it was found that the RDH possesses an AB/HR of approximately 3.16, well under the best baseball players which average at around 10. Next, it was found that while an increase in fence height increases the AB/HR ratio of the RDH, evident in figure 2, there is still an element of randomness due to the random distribution of initial conditions which the simulation pulls from. To achieve an AB/HR ratio of greater than 10 for the RDH, a fence height of 14.0 meters should be used. Therefore, unless a proper fence height it used, it seems as thought the RDH is far superior than any human player can be and poses a risk to the overall fairness of the sport.

References

[1] Garcia, Alejandro L. Numerical Methods for Physics (Python). CreateSpace Independent Publishing Platform, 2017.