Simulating the Trajectory of a Soccer Ball in the Presence of Air Resistance and Wind

Hannah King

October 25, 2022

1. Introduction

Simulation of the trajectory of objects on Earth has been a notoriously difficult subject for scientists to tackle given the imperfect nature of our world with the existence of wind and air resistance, for example. Through this project I hope to tackle this issue through an altered trajectory simulation using a combination of the forward derivative formula and drag force equation. This method will then be used to analyze typical soccer ball launch angles and velocities we see in real-game passes and kicks as in the famed "cucchiaio" or "spoon" penalty kick of Francesco Totti in the 2000 UEFA European Football Championship [1].

Section 2 describes a detailed explanation of my methodology in crafting my simulation code including the decomposition of accelerations, velocities, and positions into x- and y- components, the effect of drag force on acceleration, the effect of wind on drag force, the extraction of maximum height and range through nested while loops, and, of course, the forward derivative formula and the drag force equation as a basis. In section 3, I present the results of these numerical methods, and in the final section are included any final remarks discussing the significance of these results, limitations of my method, and possible next steps.

2. Methods

2a. The Forward Derivative Formula

We know that velocity is the derivative of the position at time t, and the acceleration is the second derivative. Therefore, it is necessary we construct a formula that allows us to numerically calculate the derivative. Here, we may use the definition of derivative:

$$\frac{df}{st} = \lim_{\Delta t \to 0} \frac{f(t+\Delta t)-f(t)}{\Delta t} \quad (1)$$

The limit as Δt approaches 0 of $f(t)$ can then be approximated when Δt is small and close to 0, we will name this value h:

$$\frac{df}{dt} \simeq \frac{f(t+h)-f(t)}{h} \tag{2}$$

And if we define time t as an array of values where $t_i = t_{i-1} + h$ through the process of discretization we can rewrite the above equation as

$$f'(t_i) = \frac{f(t_{i+1})-f(t_i)}{h} \rightarrow f(t_{i+1}) = f(t_i) + f'(t_i) * h \tag{3}$$

This equation is called the forward derivative formula and gives a way for us to model a function at any time t based on its previous values and the derivative at that point.

We can now illustrate the position at any time using the forward derivative formula being applied twice, once to the velocity and once to the position:

$$v(t_{i+1}) = v(t_i) + a(t_i) * h \tag{4}$$

$$s(t_{i+1}) = s(t_i) + v(t_i) * h \tag{5}$$

When air resistance is considered, the acceleration of the object also changes in a way which we will explore later. But for now, this yields an equation that we may use to find the position of an object in motion as a function of t. This equation, however, can only be used for objects moving in one dimension i.e. objects in freefall or traveling along a flat surface. A soccer ball moves both vertically and horizontally, so we need to modify these equations to incorporate this fact.

This requires we "break down" the original velocity vector applied to the soccer ball into its x and y components which we can do through trigonometric ratios:

$$cos\Theta = \frac{v_x}{v} \rightarrow v_x = vcos\Theta \tag{6}$$

$$sin\Theta = \frac{v_y}{v} \rightarrow v_y = vsin\Theta \tag{7}$$

Because movement in the horizontal and vertical axes are independent of each other, we can then explore the trajectory of the soccer ball in terms of its x and y components:
In the x direction:

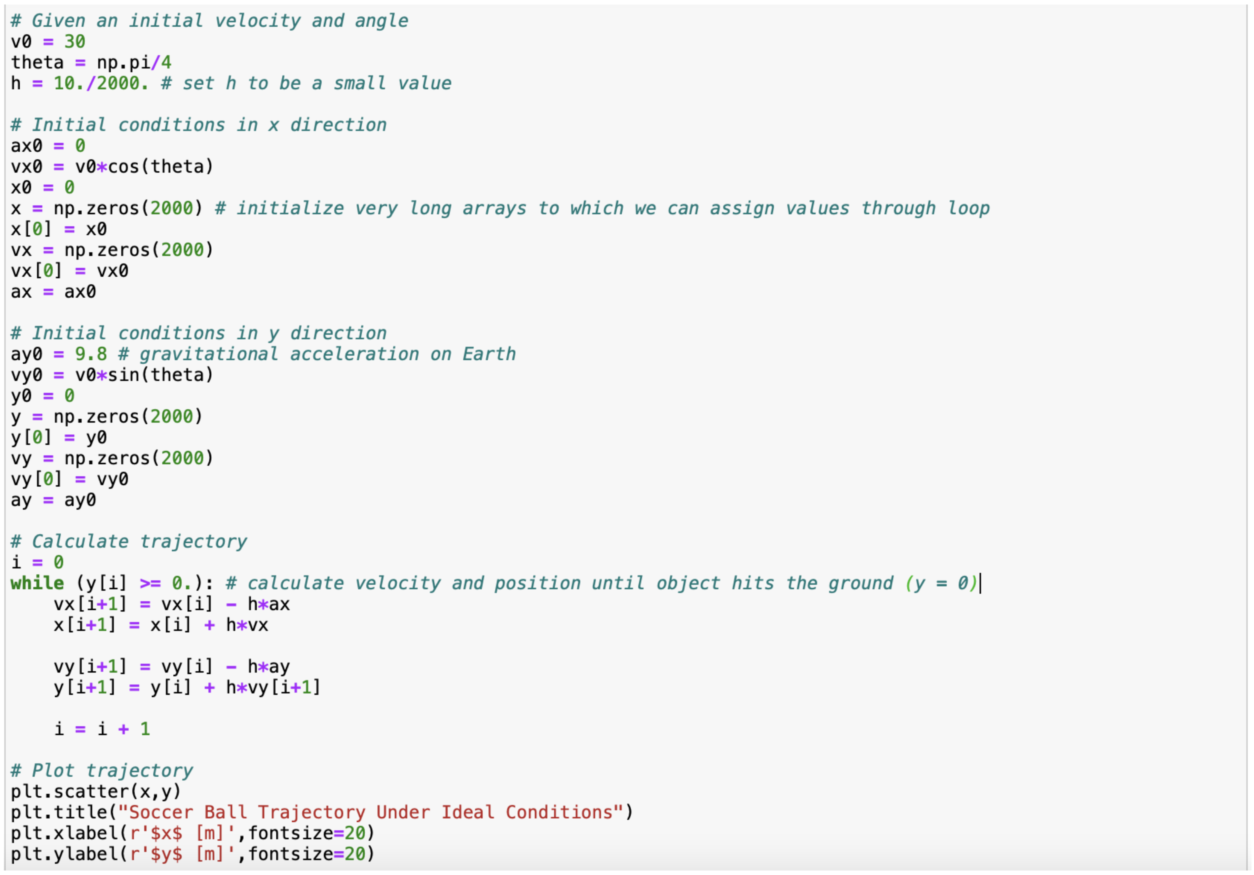
$$v_x(t_{i+1}) = v_x(t_i) + a_x(t_i) * h \tag{8}$$

$$x(t_{i+1}) = x(t_i) + v_x(t_i) * h \tag{9}$$

In the y direction:

$$v_y(t_{i+1}) = v_y(t_i) + a_y(t_i) * h \tag{10}$$

$$y(t_{i+1}) = y(t_i) + v_y(t_i) * h \tag{11}$$



```python
# Given an initial velocity and angle
v0 = 30
theta = np.pi/4
h = 10./2000. # set h to be a small value

# Initial conditions in x direction
ax0 = 0
vx0 = v0*cos(theta)
x0 = 0
x = np.zeros(2000) # initialize very long arrays to which we can assign values through loop
x[0] = x0
vx = np.zeros(2000)
vx[0] = vx0
ax = ax0

# Initial conditions in y direction
ay0 = 9.8 # gravitational acceleration on Earth
vy0 = v0*sin(theta)
y0 = 0
y = np.zeros(2000)
y[0] = y0
vy = np.zeros(2000)
vy[0] = vy0
ay = ay0

# Calculate trajectory
i = 0
while (y[i] >= 0.): # calculate velocity and position until object hits the ground (y = 0)
    vx[i+1] = vx[i] - h*ax
    x[i+1] = x[i] + h*vx

    vy[i+1] = vy[i] - h*ay
    y[i+1] = y[i] + h*vy[i+1]

    i = i + 1

# Plot trajectory
plt.scatter(x,y)
plt.title("Soccer Ball Trajectory Under Ideal Conditions")
plt.xlabel(r'$x$ [m]',fontsize=20)
plt.ylabel(r'$y$ [m]',fontsize=20)
```

Figure 1. Python implementation of equations 9 and 11 to find the trajectory of an object given an initial velocity, launch angle, and position

Fig.1 implements equations 9 and 11 using a while loop to find and plot the trajectory of an object under ideal conditions ($a_x = 0, a_y = g = 9.8\,\frac{m}{s^2}$).

2b. Incorporating Air Resistance

However, we know that acceleration is not constant in the real world because air has mass and an object moving through something with mass will obviously be slowed down. We can estimate this force of this slow-down through the drag force equation:

$$F_{drag} = -\frac{1}{2}C\rho A v^2 \hat{v} \tag{12}$$

where C is the drag coefficient, ρ is the density of air, A is the cross-sectional area of the object, v is the velocity, and $\hat{v}$ is the unit vector of the velocity vector found by the equation:

$$\hat{v} = \frac{v}{|v|} = \frac{v}{\sqrt{v_x^2 + v_y^2}} \tag{13}$$

And similarly to how we separated the components of acceleration, velocity, and position into x and y components, we can break down the drag force equation into 2, albeit in a more complicated fashion:

$$(F_{drag})_x = -\frac{1}{2}C\rho A v^2 * \frac{v_x}{\sqrt{{v_x}^2+{v_y}^2}} \tag{14}$$

We can then replace substitute $|v|$ for v:

$$(F_{drag})_x = -\frac{1}{2}C\rho A\left(\sqrt{{v_x}^2 + {v_y}^2}\right)^2 * \frac{v_x}{\sqrt{{v_x}^2+{v_y}^2}} \tag{15}$$

From here we simplify:

$$(F_{drag})_x = -\frac{1}{2}C\rho A({v_x}^2 + {v_y}^2) * \frac{v_x\sqrt{{v_x}^2+{v_y}^2}}{{v_x}^2+{v_y}^2} \tag{16}$$

$$(F_{drag})_x = -\frac{1}{2}C\rho A v_x\sqrt{{v_x}^2 + {v_y}^2} \tag{17}$$

$$(F_{drag})_x = -\frac{1}{2}C\rho A v v_x \tag{18}$$

This yields our final equation for drag force in the x direction using the fact that the velocity vector is equal to the sum of the square of its x and y components. Using the same process for the y direction yields:

$$(F_{drag})_y = -\frac{1}{2}C\rho A v v_y \tag{19}$$

Equations 18 and 19 yield the drag force as a function of the initial velocity and the velocity component, which changes. To incorporate this into our method, however, we need to show how the acceleration changes. This is relatively simple. Based on Newton's Second Law, we see that force is equal to mass multiplied by acceleration, so we may rewrite the acceleration caused by drag for x and y to be:

$$(a_{drag})_x = -\frac{1}{2m}C\rho A v v_x \tag{20}$$

$$(a_{drag})_y = -\frac{1}{2m}C\rho A v v_y \tag{21}$$

where m is equal to the mass of the projectile

Therefore, based on the fact that the acceleration in the x direction is 0 under ideal conditions and acceleration in the y direction is $g = 9.8$, the summed acceleration in x and y becomes:

$$a_x = (a_{drag})_x = -\frac{1}{2m}C\rho A v v_x \tag{22}$$

$$a_y = 9.8 - (a_{drag})_y = 9.8 + \frac{1}{2m}C\rho A v v_y \tag{23}$$

As a final step, we discretize the above equations, so that we may loop them in our code:
In the x direction:

$$a_x(t_{i+1}) = -\frac{1}{2m}C\rho A v_0 v_x(t_i) \tag{24}$$

$$v_x(t_{i+1}) = v_x(t_i) + a_x(t_i) * h$$
$$x(t_{i+1}) = x(t_i) + v_x(t_i) * h$$

And in the y direction:

$$a_y(t_{i+1}) = -\frac{1}{2m} C\rho A v_0 v_y(t_i) \tag{25}$$
$$v_y(t_{i+1}) = v_y(t_i) + a_y(t_i) * h$$
$$y(t_{i+1}) = y(t_i) + v_y(t_i) * h$$

```python
from matplotlib import pyplot as plt
import numpy as np

# constants
d = 0.22; # diameter
A = np.pi*(d/2)**2 # area of a circle
m = 0.420; # mass
ρ = 1.3; # density of air
dragcoefficient = 0.25
g = 9.8
h = 10./2000.    # Incremental step

# Input values
v0 = float(input('Initial velocity (m/s): '))
theta = float(input('Launch angle (degrees): '))*2*np.pi/360

# Initial conditions in x direction
x0 = 0
vx0 = v0*np.cos(theta)
ax0 = 0
x = np.zeros(2000)
x[0] = x0
vx = np.zeros(2000)
vx[0] = vx0
ax = np.zeros(2000) # as acceleration is no longer constant, we must also create an array for it
ax[0] = ax0

# Initial conditions in y direction
y0 = 0
vy0 = v0*np.sin(theta)
ay0 = g
y = np.zeros(2000)
y[0] = y0
vy = np.zeros(2000)
vy[0] = vy0
ay = np.zeros(2000)
ay[0] = ay0
```

```python
# Calculate trajectory
i = 0
while(y[i] >= 0.):
    Fdragx = (-1/2)*dragcoefficient*ρ*A*vx[i]*v0 # calculate drag force
    adragx = Fdragx/m # calculate acceleration caused by drag
    ax[i+1] = -adragx # total acceleration
    vx[i+1] = vx[i] - h*ax[i+1]
    x[i+1] = x[i] + h*vx[i+1]

    Fdragy = (-1/2)*dragcoefficient*ρ*A*vy[i]*v0
    adragy = Fdragy/m
    ay[i+1] = ay0 - adragy
    vy[i+1] = vy[i] - h*ay[i+1]
    y[i+1] = y[i] + h*vy[i+1]

    i = i + 1

# Plot trajectory
plt.scatter(x,y)
plt.title("Soccer Ball Trajectory with Drag Force")
plt.xlabel(r'$x$ [m]',fontsize=20)
plt.ylabel(r'$y$ [m]',fontsize=20)
```

Figure 2: Python implementation of a soccer ball trajectory with air resistance

Fig.2 implements equations 22 through 29 using a while loop to find and plot the trajectory of an object with drag force (air resistance). In terms of the constants, the drag coefficient was pulled from the webpage "Drag on a Soccer Ball," the density of air was given to be $1.3\ \frac{kg}{m^3}$, the cross-sectional area was obtained by using the diameter and understanding the spherical soccer ball's cross-section is always a circle, and the mass was also given as $0.420\ kg$ [2].

2c. Incorporating Wind

The final step to create my simulation is to incorporate the effect of wind on the trajectory. For the sake of simplicity, the wind is assumed to be moving at a constant speed in a constant direction within the $x - y$ plane. Wind is not a force, so it does not have a direct effect on the acceleration of the projectile, but it does affect the velocity of the projectile at any time which indirectly changes how we need to calculate the drag force and thus the acceleration. This fact can be expressed in equation-form as follows:

$$v_{total} = v + v_{wind} \tag{26}$$

And broken down into x and y components using trigonometry and the angle of the wind:

$$(v_x)_{total} = v_x + (v_x)_{wind} = v_x + v_{wind} cos\Theta \tag{27}$$

$$(v_y)_{total} = v_y + (v_y)_{wind} = v_y + v_{wind} sin\Theta \tag{28}$$

We may then substitute these equations for v_x and v_y in the drag force equation which yields:

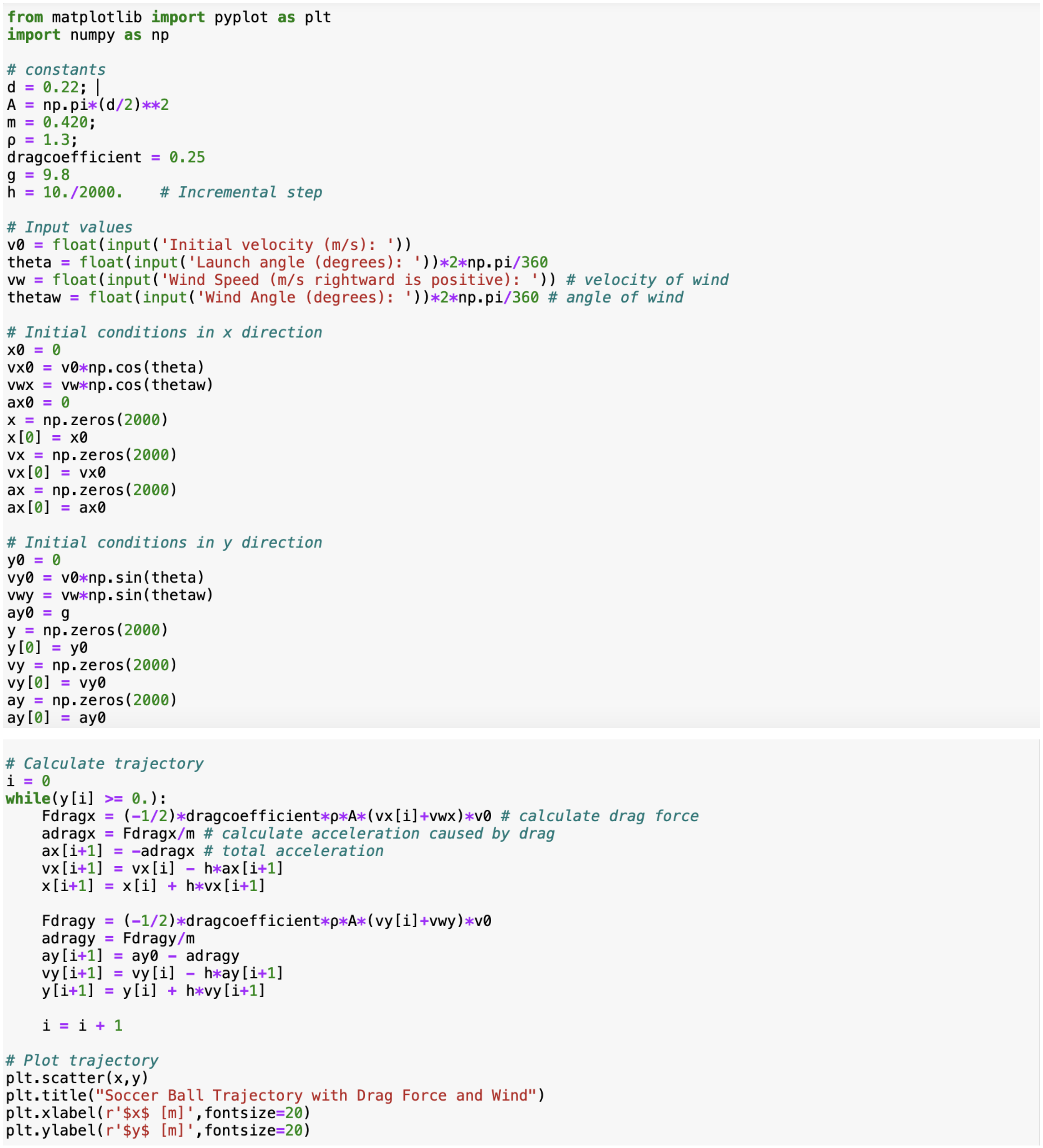
$$a_x = -\frac{1}{2m} C\rho A v(v_x + (v_x)_{wind}) \tag{29}$$

$$a_y = 9.8 + \frac{1}{2m} C\rho A v(v_y + (v_y)_{wind}) \tag{30}$$

And after discretization:

$$a_x(t_{i+1}) = -\frac{1}{2m} C\rho A v_0(v_x(t_i) + (v_x)_{wind}) \tag{31}$$

$$a_y(t_{i+1}) = -\frac{1}{2m} C\rho A v_0(v_y(t_i) + (v_y)_{wind}) \tag{32}$$



```python
from matplotlib import pyplot as plt
import numpy as np

# constants
d = 0.22; |
A = np.pi*(d/2)**2
m = 0.420;
ρ = 1.3;
dragcoefficient = 0.25
g = 9.8
h = 10./2000.    # Incremental step

# Input values
v0 = float(input('Initial velocity (m/s): '))
theta = float(input('Launch angle (degrees): '))*2*np.pi/360
vw = float(input('Wind Speed (m/s rightward is positive): ')) # velocity of wind
thetaw = float(input('Wind Angle (degrees): '))*2*np.pi/360 # angle of wind

# Initial conditions in x direction
x0 = 0
vx0 = v0*np.cos(theta)
vwx = vw*np.cos(thetaw)
ax0 = 0
x = np.zeros(2000)
x[0] = x0
vx = np.zeros(2000)
vx[0] = vx0
ax = np.zeros(2000)
ax[0] = ax0

# Initial conditions in y direction
y0 = 0
vy0 = v0*np.sin(theta)
vwy = vw*np.sin(thetaw)
ay0 = g
y = np.zeros(2000)
y[0] = y0
vy = np.zeros(2000)
vy[0] = vy0
ay = np.zeros(2000)
ay[0] = ay0
```

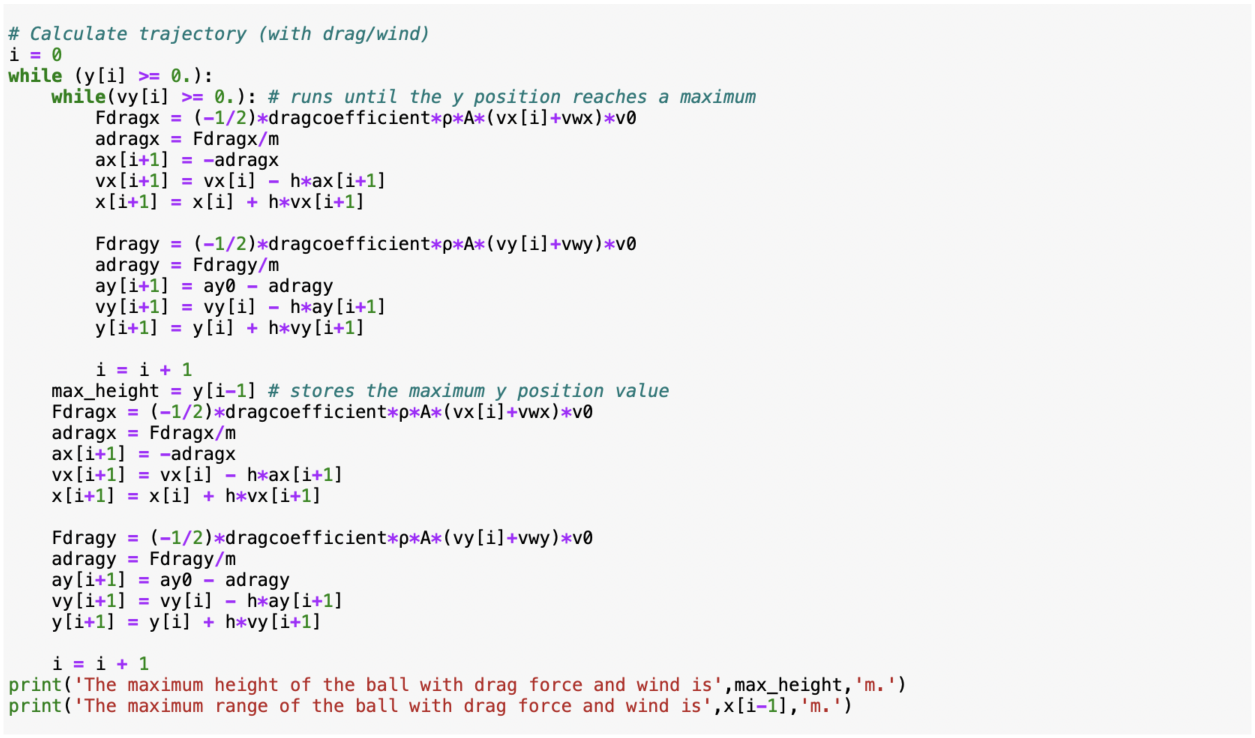
```python
# Calculate trajectory
i = 0
while(y[i] >= 0.):
    Fdragx = (-1/2)*dragcoefficient*ρ*A*(vx[i]+vwx)*v0 # calculate drag force
    adragx = Fdragx/m # calculate acceleration caused by drag
    ax[i+1] = -adragx # total acceleration
    vx[i+1] = vx[i] - h*ax[i+1]
    x[i+1] = x[i] + h*vx[i+1]

    Fdragy = (-1/2)*dragcoefficient*ρ*A*(vy[i]+vwy)*v0
    adragy = Fdragy/m
    ay[i+1] = ay0 - adragy
    vy[i+1] = vy[i] - h*ay[i+1]
    y[i+1] = y[i] + h*vy[i+1]

    i = i + 1

# Plot trajectory
plt.scatter(x,y)
plt.title("Soccer Ball Trajectory with Drag Force and Wind")
plt.xlabel(r'$x$ [m]',fontsize=20)
plt.ylabel(r'$y$ [m]',fontsize=20)
```

Figure 3: Python implementation of the trajectory algorithm considering air resistance and constant wind

Fig.3 shows the trajectory of the soccer ball considering both air resistance and wind and represents the final algorithm we may now use to calculate initial launch angles and velocities in different situations.

2d. Calculating Maximum Range and Height

Before looking into such simulations, however, I decided to experiment with this code to determine the maximum range and height of the trajectory of the soccer ball

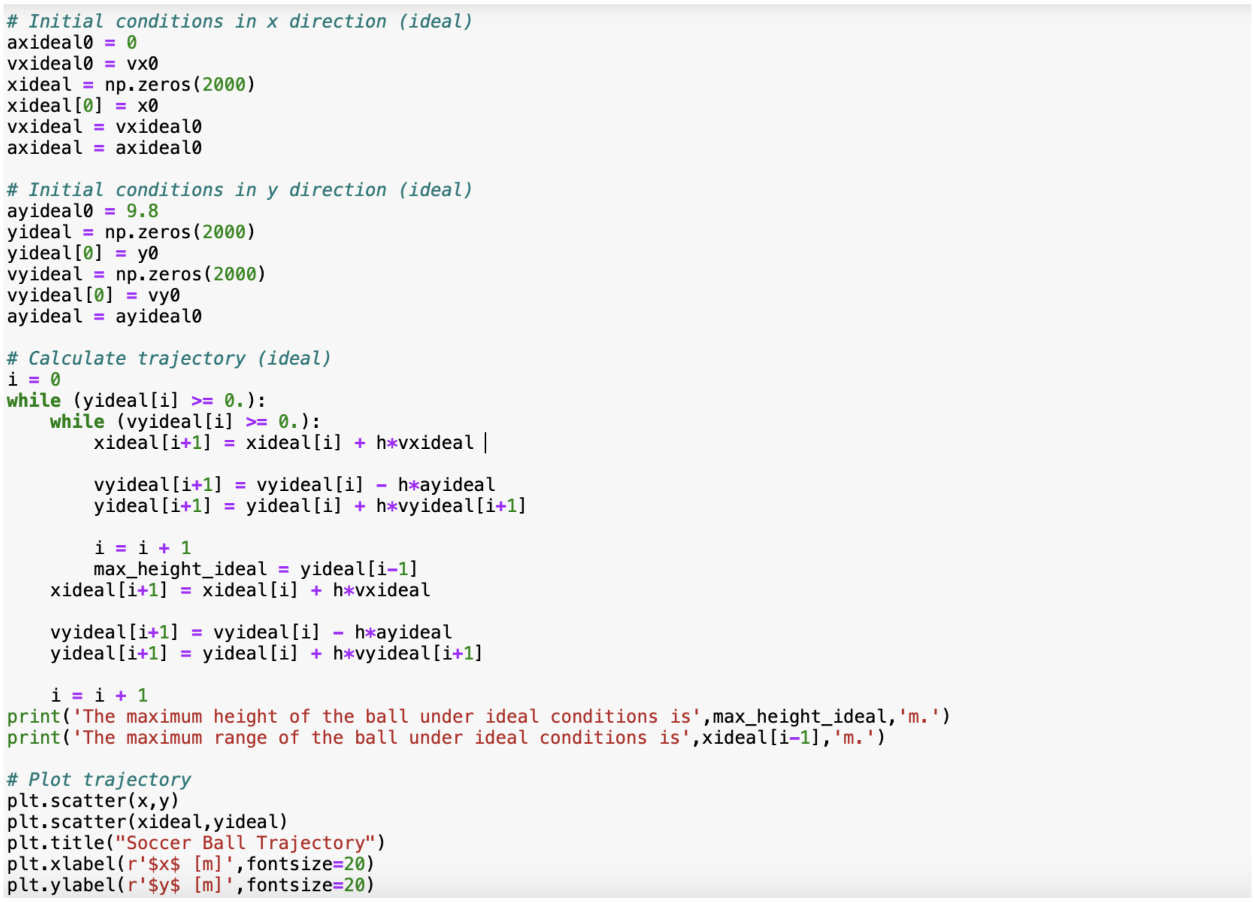


```python
# Calculate trajectory (with drag/wind)
i = 0
while (y[i] >= 0.):
    while(vy[i] >= 0.): # runs until the y position reaches a maximum
        Fdragx = (-1/2)*dragcoefficient*ρ*A*(vx[i]+vwx)*v0
        adragx = Fdragx/m
        ax[i+1] = -adragx
        vx[i+1] = vx[i] - h*ax[i+1]
        x[i+1] = x[i] + h*vx[i+1]

        Fdragy = (-1/2)*dragcoefficient*ρ*A*(vy[i]+vwy)*v0
        adragy = Fdragy/m
        ay[i+1] = ay0 - adragy
        vy[i+1] = vy[i] - h*ay[i+1]
        y[i+1] = y[i] + h*vy[i+1]

        i = i + 1
    max_height = y[i-1] # stores the maximum y position value
    Fdragx = (-1/2)*dragcoefficient*ρ*A*(vx[i]+vwx)*v0
    adragx = Fdragx/m
    ax[i+1] = -adragx
    vx[i+1] = vx[i] - h*ax[i+1]
    x[i+1] = x[i] + h*vx[i+1]

    Fdragy = (-1/2)*dragcoefficient*ρ*A*(vy[i]+vwy)*v0
    adragy = Fdragy/m
    ay[i+1] = ay0 - adragy
    vy[i+1] = vy[i] - h*ay[i+1]
    y[i+1] = y[i] + h*vy[i+1]

    i = i + 1
print('The maximum height of the ball with drag force and wind is',max_height,'m.')
print('The maximum range of the ball with drag force and wind is',x[i-1],'m.')
```

Figure 4: Python implementation of nested while loop to find the maximum height and range

under the conditions of air resistance and wind as well as under ideal conditions. Maximum range was easy to calculate because the current while loop stops when the soccer ball's y position is less than zero. Taking the x position at one h before this yields the distance traveled while the soccer ball was above the ground and represents a good estimate of the range. Maximum height occurs at some time in the middle, so it is a bit more difficult to find. However, it is known that the maximum of the position can be found by finding when the derivative, velocity, is equal to zero. Therefore, I implemented a while loop within my larger loop that ran while the y component of velocity was greater than zero. At which point this loop stopped running, the y position was at a maximum, so I created a new variable "max_height" to which I assigned the y position at one h before. This stored the value without stopping the larger while loop that ran while the y position was negative and is illustrated in Fig.4.

While not necessarily, I found it might be helpful to plot the trajectory of the soccer ball under ideal conditions as well. This code is identical to the code shown in Fig.1 but with different names for the variables. The only difference is I also implemented my nested while loop method to extract the maximum range and height under ideal conditions which I could then directly compare with the values I got from Fig.4. The implementation of this is shown in the following Fig.5. It must be noted that I removed the array of the x component of velocity because the acceleration is 0, and it is thus constant.



```
# Initial conditions in x direction (ideal)
axideal0 = 0
vxideal0 = vx0
xideal = np.zeros(2000)
xideal[0] = x0
vxideal = vxideal0
axideal = axideal0

# Initial conditions in y direction (ideal)
ayideal0 = 9.8
yideal = np.zeros(2000)
yideal[0] = y0
vyideal = np.zeros(2000)
vyideal[0] = vy0
ayideal = ayideal0

# Calculate trajectory (ideal)
i = 0
while (yideal[i] >= 0.):
    while (vyideal[i] >= 0.):
        xideal[i+1] = xideal[i] + h*vxideal

        vyideal[i+1] = vyideal[i] - h*ayideal
        yideal[i+1] = yideal[i] + h*vyideal[i+1]

        i = i + 1
        max_height_ideal = yideal[i-1]
    xideal[i+1] = xideal[i] + h*vxideal

    vyideal[i+1] = vyideal[i] - h*ayideal
    yideal[i+1] = yideal[i] + h*vyideal[i+1]

    i = i + 1
print('The maximum height of the ball under ideal conditions is',max_height_ideal,'m.')
print('The maximum range of the ball under ideal conditions is',xideal[i-1],'m.')

# Plot trajectory
plt.scatter(x,y)
plt.scatter(xideal,yideal)
plt.title("Soccer Ball Trajectory")
plt.xlabel(r'$x$ [m]',fontsize=20)
plt.ylabel(r'$y$ [m]',fontsize=20)
```

Figure 5: Python implementation of the trajectory of the soccer ball under ideal conditions using the same initial values as Fig.4

2e. Calculating Launch Angle Given Range

After inspecting the effect of air resistance and wind on the overall trajectory of a soccer ball, it is now time to see how the range of a projectile relates to its initial launch angle. To do so we consider a very specific situation in which two soccer players are conducting a pass in which one player, Diego, kicks the ball to land on the other player's, Mia's, head. The question is at what angle must Diego kick the ball for it to land on Mia's head during its descent if she is a meters away from him. To begin, I had to find some valid values for a considering her position as outlined in Fig.6.

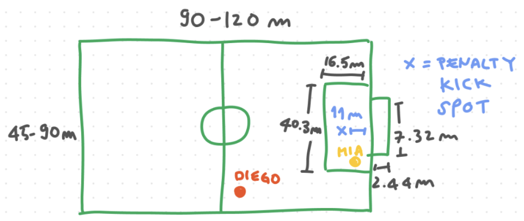
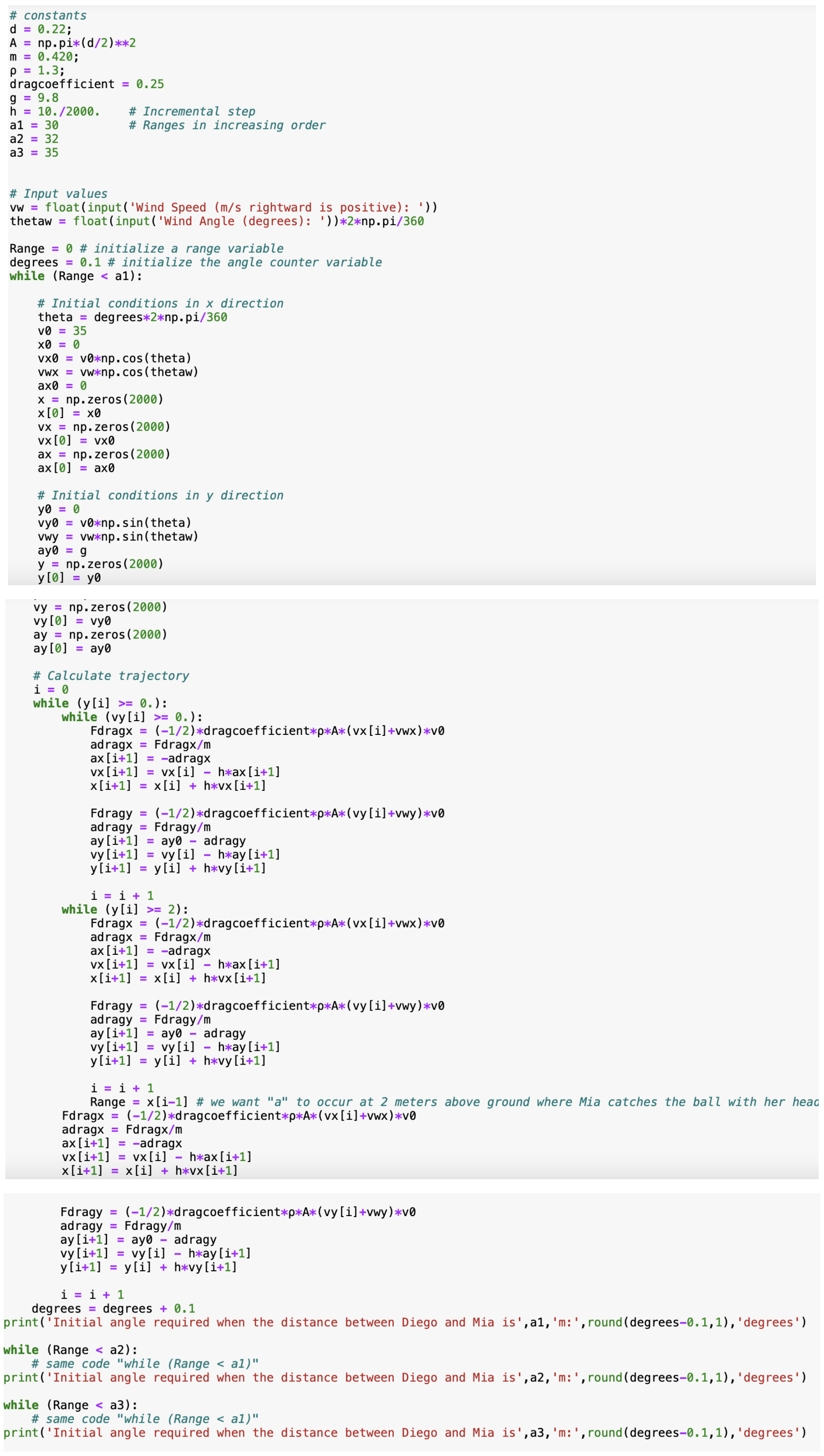


Figure 6: The relative positions of players Diego and Mia with respect to the soccer court and the goal area

Considering a half court was between $\frac{90}{2} = 45\ m$ and $\frac{120}{2} = 60\ m$, Mia and Diego spanned almost this entire length, and Mia and Diego were at an angle to each other, I assumed a to be between 30 and 40 meters. The range can only be found after employing the algorithm created in section 2a through 2d, so the next step was to loop through this algorithm using another while loop. As we are trying to determine an initial launch angle, I then initialized the launch angle (in degrees) as 0.1. Because the while loop that determined when the soccer ball touched the ground again after being kicked requires that the ball moves from the ground in the first place, it was necessary that I started the launch angle at a value greater than zero. I also created a variable "range" that would change value after every loop and initialized its value as zero. From there, I had to determine until what point I wanted the algorithm to run. Since I wanted to find the launch angle at which the target range was reached, the while loop was set to run until the variable "range" exceeded the target value which I set up as a constant. However, this range could not be set as the range after the ball once again hit the ground because Mia needed to catch the ball with her head at a position $2\ m$, given in the simulation. This required that I create a while loop that ran until the y position hit $2\ m$ in the soccer ball descent. To solve this issue, I decided to nest two while loops, one that ran while the soccer ball was moving upward ($vy > 0$) and one following that ran until the y position returned to zero. After these two loops, I had the original while loop run until the complete trajectory finished. I used this complicated method to avoid the issue of a while loop never ending as at smaller values of theta, the ball does not even reach the 2 meter mark in the first place. During these cases, only the first nested loop would run and the range value would not change. At which point the initial launch angle becomes large enough so that the maximum height of the soccer ball exceeds 2 meters, the second while loop "activates" and the range value begins to change until it reaches the target value. All that is left to do is print the value of the launch angle at one loop previous, so we are given an output to analyze.

Because I wanted to test out a few target range values, I then employed the loop two more times making sure that the range values being tested were in ascending order and the launch angle was not reset after each loop, so I could reduce computation time as each while loop did not need to start at zero. During this process, it was necessary that I pick range values that were valid i.e. achievable given the initial velocity which I decided to keep constant at $30\ \frac{m}{s}$. This required that I conduct the test with a few different range values to see whether they would converge if given a set initial velocity; these results will be elaborated upon in Section 3. Following this process yields the code seen in Fig.7 which we may use to determine launch angle as a function of the range.

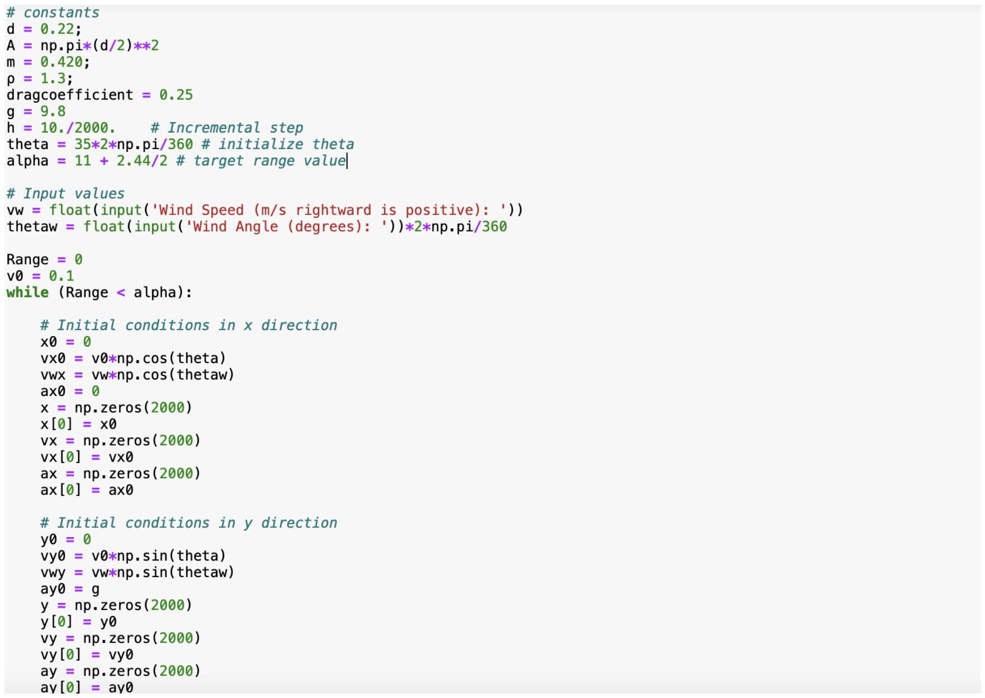


```
# constants
d = 0.22;
A = np.pi*(d/2)**2
m = 0.420;
ρ = 1.3;
dragcoefficient = 0.25
g = 9.8
h = 10./2000.    # Incremental step
a1 = 30          # Ranges in increasing order
a2 = 32
a3 = 35


# Input values
vw = float(input('Wind Speed (m/s rightward is positive): '))
thetaw = float(input('Wind Angle (degrees): '))*2*np.pi/360

Range = 0 # initialize a range variable
degrees = 0.1 # initialize the angle counter variable
while (Range < a1):

    # Initial conditions in x direction
    theta = degrees*2*np.pi/360
    v0 = 35
    x0 = 0
    vx0 = v0*np.cos(theta)
    vwx = vw*np.cos(thetaw)
    ax0 = 0
    x = np.zeros(2000)
    x[0] = x0
    vx = np.zeros(2000)
    vx[0] = vx0
    ax = np.zeros(2000)
    ax[0] = ax0

    # Initial conditions in y direction
    y0 = 0
    vy0 = v0*np.sin(theta)
    vwy = vw*np.sin(thetaw)
    ay0 = g
    y = np.zeros(2000)
    y[0] = y0
    vy = np.zeros(2000)
    vy[0] = vy0
    ay = np.zeros(2000)
    ay[0] = ay0

    # Calculate trajectory
    i = 0
    while (y[i] >= 0.):
        while (vy[i] >= 0.):
            Fdragx = (-1/2)*dragcoefficient*ρ*A*(vx[i]+vwx)*v0
            adragx = Fdragx/m
            ax[i+1] = -adragx
            vx[i+1] = vx[i] - h*ax[i+1]
            x[i+1] = x[i] + h*vx[i+1]

            Fdragy = (-1/2)*dragcoefficient*ρ*A*(vy[i]+vwy)*v0
            adragy = Fdragy/m
            ay[i+1] = ay0 - adragy
            vy[i+1] = vy[i] - h*ay[i+1]
            y[i+1] = y[i] + h*vy[i+1]

            i = i + 1
        while (y[i] >= 2):
            Fdragx = (-1/2)*dragcoefficient*ρ*A*(vx[i]+vwx)*v0
            adragx = Fdragx/m
            ax[i+1] = -adragx
            vx[i+1] = vx[i] - h*ax[i+1]
            x[i+1] = x[i] + h*vx[i+1]

            Fdragy = (-1/2)*dragcoefficient*ρ*A*(vy[i]+vwy)*v0
            adragy = Fdragy/m
            ay[i+1] = ay0 - adragy
            vy[i+1] = vy[i] - h*ay[i+1]
            y[i+1] = y[i] + h*vy[i+1]

            i = i + 1
            Range = x[i-1] # we want "a" to occur at 2 meters above ground where Mia catches the ball with her hea
        Fdragx = (-1/2)*dragcoefficient*ρ*A*(vx[i]+vwx)*v0
        adragx = Fdragx/m
        ax[i+1] = -adragx
        vx[i+1] = vx[i] - h*ax[i+1]
        x[i+1] = x[i] + h*vx[i+1]

        Fdragy = (-1/2)*dragcoefficient*ρ*A*(vy[i]+vwy)*v0
        adragy = Fdragy/m
        ay[i+1] = ay0 - adragy
        vy[i+1] = vy[i] - h*ay[i+1]
        y[i+1] = y[i] + h*vy[i+1]

        i = i + 1
    degrees = degrees + 0.1
print('Initial angle required when the distance between Diego and Mia is',a1,'m:',round(degrees-0.1,1),'degrees')

while (Range < a2):
    # same code "while (Range < a1)"
print('Initial angle required when the distance between Diego and Mia is',a2,'m:',round(degrees-0.1,1),'degrees')

while (Range < a3):
    # same code "while (Range < a1)"
print('Initial angle required when the distance between Diego and Mia is',a3,'m:',round(degrees-0.1,1),'degrees')
```

Figure 7: Python implementation of an algorithm that yields an initial launch angle given a range

2f. Calculating Initial Velocity Given Range

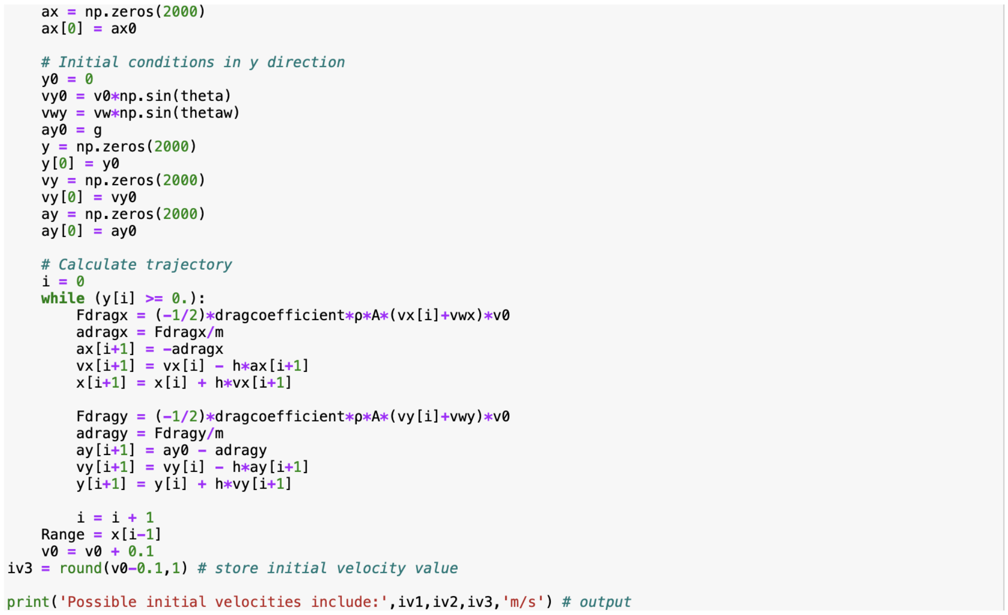
Equipped with an algorithm that is now able to find the initial conditions required for a desired range and outcome, we are able to simulate soccer player Francesco Totti's "cucchiaio" penalty kick. Looking back at footage of his penalty kick, we see that the ball moved high and slowly rather than the straight and fast trajectory we often see with shooter kicks [1]. First, we must find the relative distance between the kicker and the middle of the soccer goal where we want the ball to land slowly and softly. For this value, we may look once again at Fig.6 which shows that the penalty kicker is $11\ m$ from the front of the soccer goal and the middle of the soccer goal is another $\frac{2.44}{2} = 1.22\ m$ from the front of the soccer goal which yields a total target range of $11\ + 1.22 = 12.22\ m$. Next, we need to find our looping variable which, in this case, would be the initial velocity. For the object to have a greater height in its trajectory, we need the y component of the initial velocity to be large but not too large as to have the ending velocity be excessively large and cause a hard landing. Therefore, I set the three trial "theta" values to be 35, 40 and 45 degrees, respectively. We now have all of the values necessary to craft our algorithm that calculates initial velocity given a target range value. Because I was less confident about how the initial angle would affect the necessary initial velocity when the angle was close to the 45 degree mark as it is around this point that maximum range is reached, unlike the algorithm drawn up in Section 2e, I decided to reset the initial velocity counter and range variable, so they would run from zero. Further, I changed my range value to be the total distance the soccer ball traveled in the x direction as we were no longer considering an assist at some height above the ground. Implementation of these changes is displayed in Fig.8 below.



```
# constants
d = 0.22;
A = np.pi*(d/2)**2
m = 0.420;
ρ = 1.3;
dragcoefficient = 0.25
g = 9.8
h = 10./2000.    # Incremental step
theta = 35*2*np.pi/360 # initialize theta
alpha = 11 + 2.44/2 # target range value

# Input values
vw = float(input('Wind Speed (m/s rightward is positive): '))
thetaw = float(input('Wind Angle (degrees): '))*2*np.pi/360

Range = 0
v0 = 0.1
while (Range < alpha):

    # Initial conditions in x direction
    x0 = 0
    vx0 = v0*np.cos(theta)
    vwx = vw*np.cos(thetaw)
    ax0 = 0
    x = np.zeros(2000)
    x[0] = x0
    vx = np.zeros(2000)
    vx[0] = vx0
    ax = np.zeros(2000)
    ax[0] = ax0

    # Initial conditions in y direction
    y0 = 0
    vy0 = v0*np.sin(theta)
    vwy = vw*np.sin(thetaw)
    ay0 = g
    y = np.zeros(2000)
    y[0] = y0
    vy = np.zeros(2000)
    vy[0] = vy0
    ay = np.zeros(2000)
    ay[0] = ay0
```

```python
    # Calculate trajectory
    i = 0
    while (y[i] >= 0.):
        Fdragx = (-1/2)*dragcoefficient*ρ*A*(vx[i]+vwx)*v0
        adragx = Fdragx/m
        ax[i+1] = -adragx
        vx[i+1] = vx[i] - h*ax[i+1]
        x[i+1] = x[i] + h*vx[i+1]

        Fdragy = (-1/2)*dragcoefficient*ρ*A*(vy[i]+vwy)*v0
        adragy = Fdragy/m
        ay[i+1] = ay0 - adragy
        vy[i+1] = vy[i] - h*ay[i+1]
        y[i+1] = y[i] + h*vy[i+1]

        i = i + 1
    Range = x[i-1] # no longer need nested while loops
    v0 = v0 + 0.1
iv1 = round(v0-0.1,1) # store initial velocity value

theta = 40*2*np.pi/360 # change theta
Range = 0 # reset values
v0 = 0.1
while (Range < alpha):

    # Initial conditions in x direction
    x0 = 0
    vx0 = v0*np.cos(theta)
    vwx = vw*np.cos(thetaw)
    ax0 = 0
    x = np.zeros(2000)
    x[0] = x0
    vx = np.zeros(2000)
    vx[0] = vx0
    ax = np.zeros(2000)
    ax[0] = ax0

    # Initial conditions in y direction
    y0 = 0
    vy0 = v0*np.sin(theta)
```

```python
    vwy = vw*np.sin(thetaw)
    ay0 = g
    y = np.zeros(2000)
    y[0] = y0
    vy = np.zeros(2000)
    vy[0] = vy0
    ay = np.zeros(2000)
    ay[0] = ay0

    # Calculate trajectory
    i = 0
    while (y[i] >= 0.):
        Fdragx = (-1/2)*dragcoefficient*ρ*A*(vx[i]+vwx)*v0
        adragx = Fdragx/m
        ax[i+1] = -adragx
        vx[i+1] = vx[i] - h*ax[i+1]
        x[i+1] = x[i] + h*vx[i+1]

        Fdragy = (-1/2)*dragcoefficient*ρ*A*(vy[i]+vwy)*v0
        adragy = Fdragy/m
        ay[i+1] = ay0 - adragy
        vy[i+1] = vy[i] - h*ay[i+1]
        y[i+1] = y[i] + h*vy[i+1]

        i = i + 1
    Range = x[i-1]
    v0 = v0 + 0.1
iv2 = round(v0-0.1,1) # store initial velocity value

theta = 45*2*np.pi/360 # change theta
Range = 0
v0 = 0.1
while (Range < alpha):

    # Initial conditions in x direction
    x0 = 0
    vx0 = v0*np.cos(theta)
    vwx = vw*np.cos(thetaw)
    ax0 = 0
    x = np.zeros(2000)
    x[0] = x0
    vx = np.zeros(2000)
    vx[0] = vx0
```



```
    ax = np.zeros(2000)
    ax[0] = ax0

    # Initial conditions in y direction
    y0 = 0
    vy0 = v0*np.sin(theta)
    vwy = vw*np.sin(thetaw)
    ay0 = g
    y = np.zeros(2000)
    y[0] = y0
    vy = np.zeros(2000)
    vy[0] = vy0
    ay = np.zeros(2000)
    ay[0] = ay0

    # Calculate trajectory
    i = 0
    while (y[i] >= 0.):
        Fdragx = (-1/2)*dragcoefficient*p*A*(vx[i]+vwx)*v0
        adragx = Fdragx/m
        ax[i+1] = -adragx
        vx[i+1] = vx[i] - h*ax[i+1]
        x[i+1] = x[i] + h*vx[i+1]

        Fdragy = (-1/2)*dragcoefficient*p*A*(vy[i]+vwy)*v0
        adragy = Fdragy/m
        ay[i+1] = ay0 - adragy
        vy[i+1] = vy[i] - h*ay[i+1]
        y[i+1] = y[i] + h*vy[i+1]

        i = i + 1
    Range = x[i-1]
    v0 = v0 + 0.1
iv3 = round(v0-0.1,1) # store initial velocity value

print('Possible initial velocities include:',iv1,iv2,iv3,'m/s') # output
```

Figure 8: Python implementation of an algorithm that yields an initial velocity given a range

3. Results

In terms of the results, I will display the trajectories of the soccer ball under each algorithm outlined in each subsection of Section 2 as a scatter plot with the x position as a function of time on the x-axis and the y-position as a function of time on the y-axis.

3a. Soccer Ball Trajectory using Forward Derivative Formula and Considering Air Resistance and Wind

In order to inspect the effect that air resistance and wind has on the trajectory of the soccer ball, it was necessary that I keep the initial velocity and launch angle of the soccer ball constant between trials, which I set as 20 $\frac{m}{s}$ and 20 degrees respectively. When there is no wind, the combined Python code algorithm from Section 2c, will yield the trajectory shown in Fig.9.

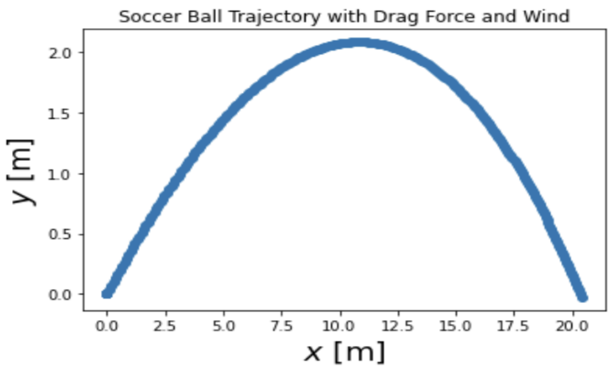
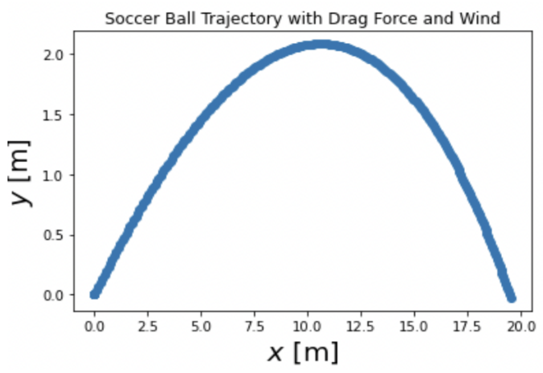
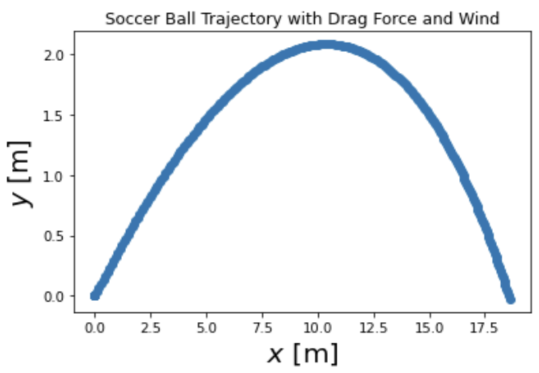


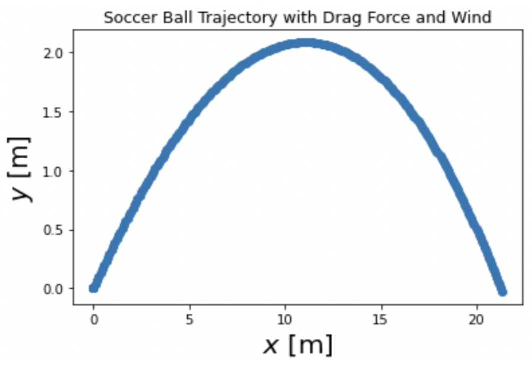
Figure 9: Trajectory of the soccer ball considering air resistance but without wind when launched at an initial velocity 20 m/s and angle 20 degrees.

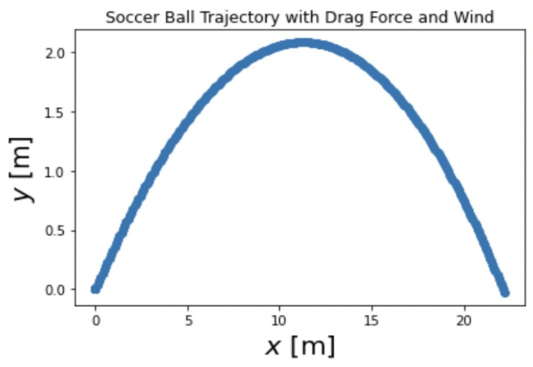




Figures 10 and 11: Trajectory of the soccer ball considering air resistance with rightward wind 4 m/s when launched at an initial velocity 20 m/s and angle 20 degrees and trajectory of the soccer ball considering air resistance with rightward wind 8 m/s when launched at an initial velocity 20 m/s and angle 20 degrees

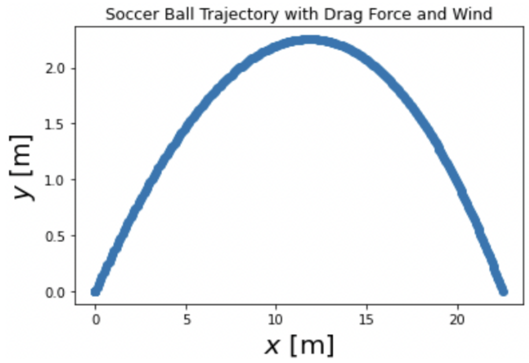
Under these conditions, it can be shown that the soccer ball travels about 20 meters and reaches a maximum height of around 2 meters. But what happens when wind is added to the mix? Figs.10 and 11 illustrate how the trajectory of the ball changes when wind is added and the magnitude of wind increases. Although the wind is moving in the same direction as the motion of the soccer ball, and the ball would, under ideal circumstances, travel further due to this boost, because there is drag force, which becomes larger as velocity increases, we see that increasing the wind speed yields a shorter range. On the other hand, when wind acts against the direction of motion as in Figs.12 and 13, the range of the soccer ball increases as there is less air resistance with the decrease in velocity.

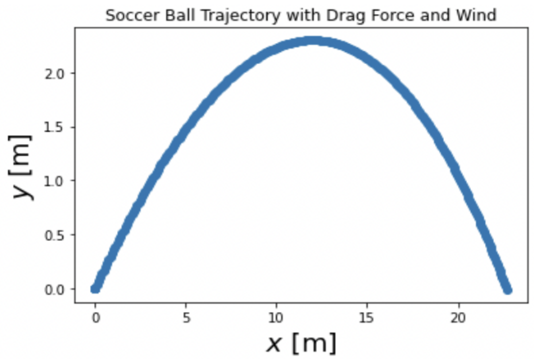




Figures 12 and 13: Trajectory of the soccer ball considering air resistance with leftward wind 4 m/s when launched at an initial velocity 20 m/s and angle 20 degrees and trajectory of the soccer ball considering air resistance with leftward wind 8 m/s when launched at an initial velocity 20 m/s and angle 20 degrees

The last variable to examine is the angle of the wind. For the sake of comparison, I decided to model the trajectory of the soccer ball given opposing wind at 45 degrees and 60 degrees below the horizontal [Fig.14,15]. Under these conditions, I saw that the range of the soccer ball grew larger as the angle of wind grew. This makes sense mathematically as well because the greater the angle, the smaller the effect of the wind on the x component of velocity. In this way, I was able to isolate each of the variables of wind to see its effect on the trajectory of the soccer ball given air resistance. I feel that it is also important to note the fact that the maximum height of the soccer ball barely changed even while changes were applied in terms of wind. This result undoubtedly shows the greater effect that wind tends to have on the horizontal motion of a projectile versus its vertical motion.





Figures 14 and 15: Trajectory of the soccer ball considering air resistance with wind 4 m/s at a 45 degree angle against the direction of the soccer ball's motion when launched at an initial velocity 20 m/s and angle 20 degrees and trajectory of the soccer ball considering air resistance with wind 4 m/s at a 60 degree angle against the direction of the soccer ball's motion when launched at an initial velocity 20 m/s and angle 20 degrees

Now, the question may arise of how the trajectory has been affected by just the incorporation of air resistance; this is something that we may examine using the algorithm created in Section 2d that shows the difference in trajectory and thus maximum height and range of a soccer ball launched under the effect of air resistance and without. The output of this code can be seen in Fig.16. The trajectory of the soccer ball with drag force understandably has both a smaller range and smaller maximum height than the trajectory of the soccer ball under ideal conditions. It is also interesting to note that the trajectory of the soccer ball with air resistance is no longer perfectly parabolic like the trajectory of the ball under ideal conditions meaning its maximum occurs slightly after the halfway point of the range.

```
The maximum height of the ball with drag force and wind is 4.106812577641195 m.
The maximum range of the ball with drag force and wind is 35.406279756609095 m.
The maximum height of the ball under ideal conditions is 5.34580649325972 m.
The maximum range of the ball under ideal conditions is 58.777773430158916 m.

Text(0, 0.5, '$y$ [m]')
```

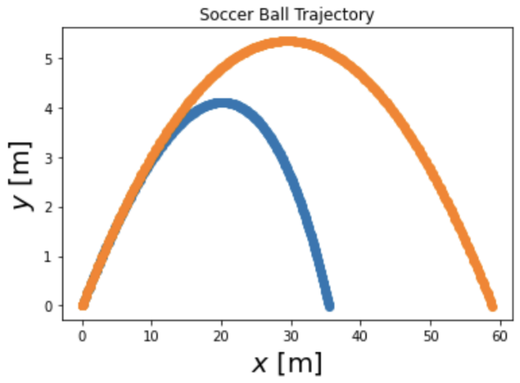
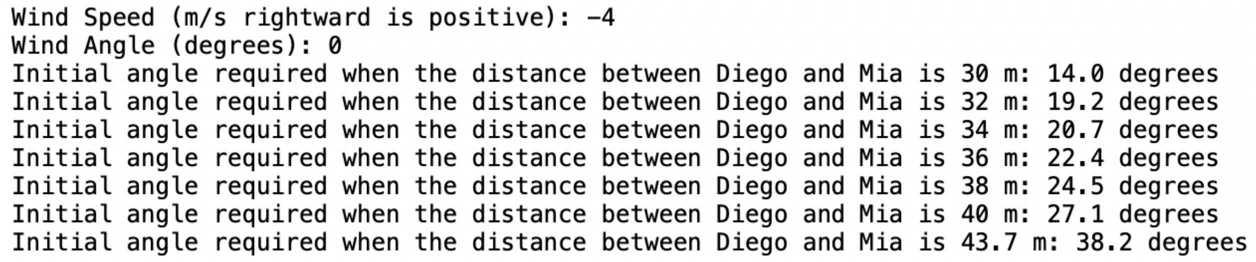


Figure 16: Trajectory of a soccer ball with air resistance vs under ideal conditions given no wind and an initial velocity 30 m/s and launch angle 20 degrees

3b. Possible Initial Launch Angles Given Range Values



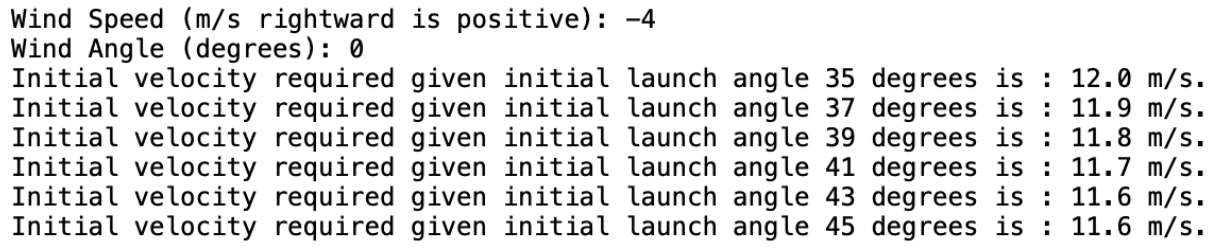
```
Wind Speed (m/s rightward is positive): -4
Wind Angle (degrees): 0
Initial angle required when the distance between Diego and Mia is 30 m: 14.0 degrees
Initial angle required when the distance between Diego and Mia is 32 m: 19.2 degrees
Initial angle required when the distance between Diego and Mia is 34 m: 20.7 degrees
Initial angle required when the distance between Diego and Mia is 36 m: 22.4 degrees
Initial angle required when the distance between Diego and Mia is 38 m: 24.5 degrees
Initial angle required when the distance between Diego and Mia is 40 m: 27.1 degrees
Initial angle required when the distance between Diego and Mia is 43.7 m: 38.2 degrees
```

Figure 17: The initial angle required for range values between 30 and 40 meters taken at 2 meter intervals and at 43.7 meters when there is a leftward wind of 4 m/s

Next I would like to discuss the results of my Python code implementation of the methods described in Section 2e in which I created a function of launch angle given a target range value. And based on my previous assumption that this range value for the given situation was likely between 30 and 40 meters and the fact that professional soccer players kick the ball at speeds around $30\ \frac{m}{s}$, I was able to create a few possible trajectories and initial launch angles which are represented as the output in Fig.17. Given the arbitrarily chosen wind velocity of $-\ 4\ \frac{m}{s}$, the soccer ball would need to be kicked at an angle within the approximate range 14 to 27.1 degrees. I also included the angle at the point where maximum range was achieved which was 38.2 degrees. Based on prior physics knowledge, under ideal conditions, maximum range occurs when the launch angle is 45 degrees, so I found it interesting that the incorporation of drag forces and wind would decrease this optimal launch angle to about half this number.

3c. Possible Initial Launch Velocities Given Launch Angle

Finally come the results of what initial launch velocities are required to recreate the "cucchiaio" kick given the positioning described in Section 3f [Fig.18]. In terms of constants, the distance between the penalty kick location and the middle of the goal was 12.22 meters and the range of initial launch angles to achieve a high and slow trajectory was estimated to be between 35 and 45 degrees. After seeing the results from Section 3b, I kept this range as it perfectly straddled the angle at which maximum range was achieved. This range in initial launch angle yielded possible initial launch velocities between 12.0 and 11.6 m/s. This value is significantly less than the 30 m/s that was used



```
Wind Speed (m/s rightward is positive): -4
Wind Angle (degrees): 0
Initial velocity required given initial launch angle 35 degrees is : 12.0 m/s.
Initial velocity required given initial launch angle 37 degrees is : 11.9 m/s.
Initial velocity required given initial launch angle 39 degrees is : 11.8 m/s.
Initial velocity required given initial launch angle 41 degrees is : 11.7 m/s.
Initial velocity required given initial launch angle 43 degrees is : 11.6 m/s.
Initial velocity required given initial launch angle 45 degrees is : 11.6 m/s.
```

Figure 18: The initial launch velocity required for a target range 12.22 meters given an initial launch angle between 35 and 45 degrees taken at 2 degree intervals when there is a leftward wind of 4 m/s

for the velocity with which soccer players tend to kick the ball. As a rule of thumb, as the launch angle increases, the required initial velocity decreases and creates a slower and higher trajectory.

4. Discussion and Conclusion

Ultimately this project allowed me to deduce that the famous “cucchiaio” kick requires kick speeds significantly less than those employed generally during on-field play (11 m/s for the “cucchiaio” and 30 m/s for on-field). This discovery is significant as it shows that a nuanced combination of launch angle and speed is required to achieve the penalty kick as Francesco Totti did considering the shorter distance between the penalty kick position and the goal. To end, I would like to address that my methodology is far from perfect and fails to address many of the caveats that arise with the imperfect nature of motion on our planet. One major weakness of this system is that it is unable to consider wind that is not constant or occurs outside of the flat, two-dimensional plane through which we examined motion. Indeed, there exists a separate z axis in our three-dimensional world which would greatly affect the final position of our soccer ball as well as its general trajectory. That being said, this doesn’t change the fact that my final algorithm serves as a good estimate of required launch angles and launch velocities to yield a specific range. This algorithm could be improved and used by soccer players to improve their performance through a strictly scientific lens which often yields much more reliable results than mere trial and error.

References

[1] “Francesco Totti il ‘cucchiaio’, commento Pizzul.” *YouTube*, uploaded by Ford Cosworth 1973, 7 April 2018, https://www.youtube.com/watch?v=XsKr4FX9y0w.

[2] Hall, Nancy. “Drag on a Soccer Ball.” *NASA*, 13 May 2021, https://www.grc.nasa.gov/www/k-12/airplane/socdrag.html.