Report

1. Introduction

Projectile motion is a very important field of study in physics. It has a wide array of applications from space exploration to the military to professional sports. The ability to know when or where a projectile will land is very important in these many fields. This project studies the motion of a projectile in two dimensions. Specifically, this report studies the motion of a soccer ball and the effects of drag and wind.

Section 2 will describe describe the analytical methods that will be used to solve various problems related to the soccer ball's motion. Section 3 will describe the implementation of the methods in code. Section 4 will show the results of the code and discuss the results. Section 5 will conclude the report.

2. Method

2.1: Exercise 1

Projectiles in two dimensions follow a parabolic trajectory. In a vacuum, the motion of a projectile can be described by the following equations:

$$x(t) = v_{0x}t \tag{1}$$

$$y(t) = v_{0y}t - \frac{1}{2}gt^2 \tag{2}$$

$$v_x(t) = v_{0x} \tag{3}$$

$$v_y(t) = v_{0y} - gt \tag{4}$$

$$\vec{v}(t) = v_x(t)\hat{i} + v_y(t)\hat{j} \tag{5}$$

$$v(t) = \sqrt{v_x^2 + v_y^2} \tag{6}$$

$$a_x = 0 \tag{7}$$

$$a_y = -g \tag{8}$$

where t is time at a specfic instance and t_0 is the initial time. x_0,y_0 and v_{0x},v_{0y} are the initial positions and velocities in the x and y directions, respectively, g is the acceleration due to gravity, x,y and v_x, v_y are the positions and velocities in the x and y directions at a given time, respectively. v is the magnitude of the velocity, and θ is the angle of the velocity vector with respect to the x-axis. The

vertical component of acceleration a_y is -g, similar to that of objects in free fall, while the horizontal component a_x is zero

However, in the real world, there are other forces that act on the projectile. These forces include drag force.

As an object moves through the air, the air exerts a drag force on the object. The drag force is **always in the opposite direction of** $\vec{v}$ and the force's magnitude increases as the object's speed increases. The drag force can be written as

$$\vec{F}_{drag} = \frac{1}{2} C\rho A v^2 \,(\text{in the opposite direction of } \vec{\text{v}}), \tag{9}$$

where C is the drag coefficient, ρ is the density of air, A is the cross-sectional area of the object, v is the magnitude of the velocity and m is the mass. If we introduce a unit vector in the direction of the velocity $\vec{v}$

$$\hat{v} = \frac{\vec{v}}{v} = \frac{\vec{v}}{\sqrt{v_x^2 + v_y^2}} = \frac{v_x}{\sqrt{v_x^2 + v_y^2}}\,\hat{i} + \frac{v_y}{\sqrt{v_x^2 + v_y^2}}\,\hat{j}\,, \tag{10}$$

then we can write the drag force as

$$\vec{F}_{drag} = -\frac{1}{2} C\rho A v^2\,\hat{v} \; = m\vec{a}_{drag}. \tag{11}$$

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

Furthurmore, the real world also can present wind. Wind can be represented as the following.

$$\vec{v} \longrightarrow \vec{v} - \vec{v}_{wind}\,, \tag{13}$$

$$v_x \longrightarrow v_x - (v_{wind})_x\,, \tag{14}$$

$$v_y \longrightarrow v_y - (v_{wind})_y\,. \tag{15}$$

Thus, wind's affect on $\vec{v}$ extends into the the drag force as shown in equations (10) and (11).

To find the trajectory of the soccer ball, we need to solve the following differential equations:

The trajectory of a projectile through air with wind given an initial ball velocity v_0, a launch angle θ, wind velocity v_w, and wind direction θ_w can be calculated using the following equations:

$$v_{0_x} = v_0 \cos(\theta) \tag{16}$$
$$v_{0_y} = v_0 \sin(\theta) \tag{17}$$
$$\vec{v}_{wind} = v_w \cos(\theta_w)\hat{i} + v_w \sin(\theta_w)\hat{j} \tag{18}$$
$$\vec{v}(t) = \vec{v}(t) + \vec{v}_{wind} \tag{19}$$
$$\vec{v}(t) = v_x(t) + v_y(t) + (v_{wind_x} + v_{wind_y}) \tag{20}$$
$$v(t) = \sqrt{{v_x}^2(t) + {v_y}^2(t)} \tag{21}$$
$$\hat{v}(t) = \frac{\vec{v}(t)}{v(t)} \tag{22}$$
$$\vec{F_{drag}} = \frac{1}{2} C_d \rho A v^2 (-\hat{v}) \tag{23}$$
$$\vec{F_{gravity}} = -mg\hat{j} \tag{24}$$
$$\vec{F_{net}} = \vec{F_{drag}} + \vec{F_{gravity}} \tag{25}$$
$$\vec{a} = \frac{F_{net}}{m} \tag{26}$$
$$\vec{v}(t + dt) = \vec{v}(t) + \vec{a}dt \tag{27}$$
$$v_x(t + dt) = v_x(t) + a_x(t)dt \tag{28}$$
$$v_y(t + dt) = v_y(t) + a_y(t)dt \tag{29}$$
$$\vec{r}(t + dt) = \vec{r}(t) + \vec{v}(t)dt \tag{30}$$
$$\vec{r}(t) = \tag{31}$$
$$x(t + dt) = x(t) + v_x(t)dt \tag{32}$$
$$y(t + dt) = y(t) + v_y(t)dt \tag{33}$$

Using a loop to increment through all values of t until the ball hits the ground, these equations can be implemented in `python` to calculate the trajectory of the soccer ball. Furthur discussion of the implementation of these equations can be found in section 3.1.

2.2: Exercise 2

Exercise 2 asks to write a function to calculate the maximum height and range of a soccer ball launched at a given angle with the given average professional kick velocity, $30\frac{m}{s}$.

The maximum height of the soccer ball can be found by recording the height y at time t from t_0 to t_f. The maximum height is the maximum value of $y(t)$. Similarly, the range of the soccer ball can be found by recording the position x at time t from t_0 to t_f. The range is the maximum value of $x(t)$. The maximum height and range can be found by finding the maximum value of the arrays that store the values of $y(t)$ and $x(t)$, respectively. The maximum height and range can be found by the following equations:

$$H_{max} = max(y(t)) \tag{34}$$

$$R_{max} = max(x(t)) \tag{35}$$

The implementation of these equations in python code will be explored in section 2.2.

2.3: Exercise 3

Exercise 3 asks to calculate the launch angle that Diego must kick a ball at a given velocity to his teammate Mia, who is a certain distance x_f meters away so that she may head the ball. For this exercise we will assume that there is no air resistance.

The initial velocity suggested to start at is the average professional kick velocity, $30\frac{m}{s}$. However it can be increased up to $58.6\frac{m}{s}$ Mia is 1.7m tall and can jump on average 0.3 meters.

To find the appropriate launch angle we need to find a launch angle that will give us a distance travelled of x meters at time t_f, and a height at $y(t_f) \leq 2$ meters.

To find the appropriate launch angle we can use the following equations:

$$x(t) = x_0 + v_{0_x} t \tag{36}$$

x_0 = 0, and $v_{0_x} = v_0 \cos(\theta)$, so we can simplify (36) to:

$$x(t) = v_0 \cos(\theta) t \tag{37}$$

Convert (37) to find t.

$$t = \frac{x}{v_0 \cos(\theta)} \tag{38}$$

$$y(t) = v_0 \sin(\theta) t - \frac{1}{2} g t^2 \tag{39}$$

Now substitute (37) into (38).

$$y = \frac{v_0 \sin(\theta) x}{v0 \cos(\theta)} - \frac{g x^2}{2 v_0^2 \cos^2(\theta)} \tag{40}$$

$$y = x \tan(\theta) - \frac{g x^2 \sec^2(\theta)}{2 v_0^2} \tag{41}$$

$\sec^2(\theta)$ can be written as $\tan^2(\theta) + 1$. Thus, we can simplify (41) to:

$$y = x \tan(\theta) - \frac{g x^2 (\tan^2(\theta) + 1)}{2 v_0^2} \tag{42}$$

While (42) looks intimidating, it is actually just a quadratic formula with $\tan(\theta)$ as our variable. We can simplify (42) to:

$$y = x \tan(\theta) - \frac{g x^2 \tan^2(\theta)}{2 v_0^2} - \frac{g x^2}{2 v_0^2} \tag{43}$$

If we put all variables on one side of the equation, we get:

$$0 = x\tan(\theta) - y - \frac{gx^2\tan^2(\theta)}{2v_0^2} - \frac{gx^2}{2v_0^2} \tag{44}$$

which can be expressed on the LHS as:

$$\frac{gx^2\tan^2(\theta)}{2v_0^2} - x\tan(\theta) + y + \frac{gx^2}{2v_0^2} = 0 \tag{45}$$

where $a = \frac{gx^2}{2v_0^2}$, $b = -x$, and $c = y + \frac{gx^2}{2v_0^2}$

$$0 = \frac{gx^2}{2v_0^2}\tan^2(\theta) - x\tan(\theta) - y + \frac{gx^2}{2v_0^2} \tag{46}$$

To solve for θ we can use the quadratic formula.

$$\theta = \arctan(\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}) \tag{47}$$

and plug in our values for a, b, and c.

$$\theta = \arctan(\frac{-(-x) \pm \sqrt{(-x)^2 - 4(\frac{gx^2}{2v_0^2})(y + \frac{gx^2}{2v_0^2})}}{2(\frac{gx^2}{2v_0^2})}) \tag{48}$$

Section 3.3 will discuss the implementation of these equations with real values to solve for the the launch angles, as well as provide python code and plots to back up the analytical solutions.

EDIT:

After further consideration on this question after completing sections 2.3 and 3.3, I realized that I was not accounting for the air resistance when that is the purpose of the project. Given the time constraints I am working under, I have implemented a coded solution that I believe gives the correct angle given the conditions, however it is found through brute force and a little estimation, and not actual analytical methods. The code is found in section 3.3 and the results are found in section 4.3.

2.4: Exercise 4

NOTE: I did not truly have a solution or proper approach to this question until after revising my work on exercise 3. Therefore, The results shown in **4.4** are approximations are hard coded, and certainly not analytically achieved, however, I believe they are reasonable approximations.

To estimate the a reasonable initial velocity magnitude and direction for a penalty kick to slowly land in the middle of the goal, it is important to first have reasonable parameters. The penalty marker is $11m$ from the goal line. The back of the goal is an additional $2.44m$ behind the goal line. Therefore the horizontal distance x is between 11 and 13.44 meters. We can represent this as the target distance being 12.22 and tolerance of $\pm 1.22m$.

To estimate a reasonable initial velocity magnitude and direction for a penalty kick to slowly land in the middle of the goal, we can use the functions developed in Exercise 3 to find the slowest velocity possible that can reach the goal given the current wind conditions. This can be done by looping through a range of velocities until say, the max velocity $58.6\frac{m}{s}$. The code for this is found in section 3.4 and the results are found in section 4.4.

NOTE this is a computationally complex way to find an optimal solution, nor is it a correct analytical way. However, given the time constraints I am working under, I believe this is the best solution I can provide.

3. Implementation

3.1: Exercise 1

Exercise 1 asks to write code to calculate the trajectory of a soccer ball accounting for the drag force and the wind. Exercise 1 also asks to plot the trajectory of the soccer ball with different wind speeds and directions.

To model the trajectory we need to need to find the position and velocity of the soccer ball from t_0 to t_f, when the ball hits the ground (when $y(t) = 0, t \neq 0$). In section 2.1 we describe the equations necessary to find the position and velocity of the soccer ball. We can model these equations in python to estimate the position and velocity of the soccer ball at any given time.

```
while y_graph[i] >= 0:
        rel_v = [v[0]+w[0], v[1]+w[1]]
        v_magnitude = np.sqrt(rel_v[0]**2+rel_v[1]**2)
        unit_v = rel_v/v_magnitude
        F_drag = alpha*(v_magnitude**2)*-unit_v
        F = [a[0]*mass, a[1]*mass]
        F_net = [F[0]+F_drag[0], F[1]+F_drag[1]]
        v = [v[0]+(F_net[0]/mass)*dt, v[1]+(F_net[1]/mass)*dt]
        x = x + v[0]*dt
        y = y + v[1]*dt
        x_graph[i+1] = x
        y_graph[i+1] = y
        i+=1
```

The code above is the main loop of the program. The loop runs until the soccer ball hits the ground. The loop calculates the relative velocity of the soccer ball, the magnitude of the relative velocity, the unit vector of the relative velocity, the drag force, the net force, the velocity, and the position. The loop then updates the position and velocity of the soccer ball and the index of the arrays that store the position and velocity of the soccer ball. This loop stores the positions used for the graph

separately from the values in the loop so that the graph can be plotted without the graph being updated every time the loop runs.

The code below is the code that plots the graph of the trajectory of the soccer ball.

```
w = wind(wind_velocity,wind_angle)
    plt.figure(figsize=(10,10))
    x_graph,y_graph = soccer_ball_trajectory(x0,y0,v0, launch_angle, ball_and_environme

    x_ceiling = int(np.ceil(x_graph[-1]/5)*5)
    y_ceiling = int(np.ceil(y_graph[-1]/5)*5)
    x_floor = int(np.floor(x_graph[-1]/5)*5)
    y_floor = int(np.floor(y_graph[-1]/5)*5)
    # create grid
    X = np.linspace(min(0,x_floor), max(50,x_ceiling), 5)
    Y = np.linspace(min(0,y_floor), max(40,y_ceiling), 5)
    Y, X = np.meshgrid(Y, X)
    U = np.ones_like(X)*w[0]
    V = np.ones_like(Y)*w[1]


    # plot wind vector field
    plt.quiver(X, Y, U, V, angles='xy', scale_units='xy', scale=1,
            alpha = 0.65, color = wind_color, label = f'Wind Velocity = {wind_velocity}

    # plot trajectory
    plt.plot(x_graph, y_graph, label = 'Trajectory')
    plt.xticks(np.arange(-5, max(50,x_ceiling), step=5))
    plt.yticks(np.arange(-5, max(40,40+y_ceiling), step=5))
    plt.title('Soccer Ball Trajectory with Wind')
    plt.xlim(-5, x_ceiling)
    plt.ylim(-5, max(40,y_ceiling))
    plt.xlabel('x')
    plt.ylabel('y')
    plt.axhline(y=0, color='black', linestyle='-')
    plt.axvline(x=0, color='black', linestyle='-')
    plt.legend(loc = 'best')
    plt.grid()
    plt.show()
```

3.2: Exercise 2

Exercise 2 asks to write a function to calculate the maximum height and range of a soccer ball launched at a given angle with the given average professional kick velocity, $30\frac{m}{s}$.

Before we can calculate the maximum height and range of the soccer ball, we need to calculate the trajectory of the soccer ball. We can use the code from 3.1 for the trajectory of the soccer ball with drag. However, we also need a function to calculate the trajectory of the soccer ball without drag. The code below is the function that calculates the trajectory of the soccer ball without drag, as well as the code that plots it.

```
def soccer_ball_trajectory_no_drag(
        x0 = 0,
        y0 = 0,
        v0 = 30,
        launch_angle = 30,
        ball_and_environment_properties= ball_and_environment_properties,
        ):
    # Soccer ball and environment constants
    g = ball_and_environment_properties['g']

    # Initial Conditions
    nstep = 100000
    t = np.linspace(0., 60., nstep)
    dt = t[1]-t[0]
    v = velocity(v0,launch_angle)
    a = acceleration(g)
    x = x0
    y = y0
    x_graph = np.zeros_like(t)
    y_graph = np.zeros_like(t)
    x_graph[0] = x
    y_graph[0] = y
    i = 0
    while y_graph[i] >= 0:
        v = [v[0], v[1]+a[1]*dt]
        x = x + v[0]*dt
        y = y + v[1]*dt
        x_graph[i+1] = x
        y_graph[i+1] = y
        i+=1

    x_graph = x_graph[:i]
    y_graph = y_graph[:i]
    return x_graph,y_graph

# find max range and height for launch angles between 0 and 90 degrees
launch_angles = np.linspace(0,90,91)
max_ranges_no_drag = []
max_heights_no_drag = []
for launch_angle in launch_angles:
    max_range_no_drag,max_height_no_drag = max_range_and_height_no_drag(launch_angle)
    max_ranges_no_drag.append(max_range_no_drag)
```

```
    max_heights_no_drag.append(max_height_no_drag)

# plot max range and height vs launch angle
plt.plot(launch_angles,max_ranges_no_drag, label = 'Max Range')
plt.plot(launch_angles,max_heights_no_drag, label = 'Max Height')
plt.xlabel('Launch Angle (deg)')
plt.ylabel('Distance (m)')
plt.axhline(y=0, color='black', linestyle='-')
plt.axvline(x=0, color='black', linestyle='-')
plt.xticks(np.arange(0, max(launch_angles), step=5))
plt.yticks(np.arange(0, int(np.ceil((max(max_ranges_no_drag)/5))*5)+5, step=5))
plt.xlim(0, np.ceil(max(launch_angles)/5)*5)
plt.ylim(0, int(np.ceil((max(max_ranges_no_drag)/5))*5)+5)
plt.title(f'Max Range and Height vs Launch Angle without Drag, launched at {v0} m/s')
plt.legend()
plt.grid()
plt.show()
```

 Exercise 2,Graph 1

The code below is the function that calculates the maximum height and range of a soccer ball launched at a given angle with the given average professional kick velocity, $30\frac{m}{s}$ with drag and no wind. The code to plot the model has been omitted from this report because is quite similar to that above for the model with no drag.

```
def max_range_and_height(launch_angle:float):
    x_graph,y_graph = soccer_ball_trajectory(x0,y0,v0, launch_angle, ball_and_environme
    max_range = max(np.sqrt(x_graph**2))
    max_height = max(y_graph)
    return max_range,max_height

 f#ind max range and height for launch angles between 0 and 90 degrees
launch_angles = np.linspace(0,90,91)

max_ranges = []
max_heights = []
for launch_angle in launch_angles:
    max_range,max_height = max_range_and_height(launch_angle)
    max_ranges.append(max_range)
    max_heights.append(max_height)
```

 Exercise 2,Graph 2

and below is the function that calculates the maximum height and range of a soccer ball launched at a given angle with the given average professional kick velocity, $30\frac{m}{s}$ with drag **and** wind, as well as the code to plot it. This example gives five differnt wind conditions to demonstrate it's effect. The

arrowgrid in the following plots indicate the direction and magnitude of the wind. The plotting code has been included because this plotting method is unique to the former models.

```
# find max range and height for launch angles between 0 and 90 degrees with wind
def max_range_and_height_with_wind(launch_angle:float,wind_velocity:float,wind_angle:fl
    x_graph,y_graph = soccer_ball_trajectory(x0,y0,v0, launch_angle, ball_and_environme
    #max_range = x_graph[-1]
    max_range = max(np.sqrt(x_graph**2))
    max_height = max(y_graph)
    return max_range,max_height

# find max range and height for launch angles between 0 and 90 degrees

wind_velocities = [5,1,0,1,5] # you can change this to any value
wind_angles = [180,180,0,0,0]
wind_colors = ['red','orange','green','blue','purple']
for wind_velocity,wind_angle,wind_color in zip(wind_velocities,wind_angles,wind_colors)
    w = wind(wind_velocity,wind_angle)
    plt.figure(figsize=(10,10))
    launch_angles = np.linspace(0,90,91)
    max_ranges_with_wind = []
    max_heights_with_wind = []
    for launch_angle in launch_angles:
        max_range_with_wind,max_height_with_wind = max_range_and_height_with_wind(launc
        max_ranges_with_wind.append(max_range_with_wind)
        max_heights_with_wind.append(max_height_with_wind)

    # create grid
    X = np.linspace(min(0,x_floor), max(launch_angles), 5)
    Y = np.linspace(min(0,y_floor), int(np.ceil((max(max_ranges_with_wind)/5))*5)+5, 5)
    Y, X = np.meshgrid(Y, X)
    U = np.ones_like(X)*w[0]
    V = np.ones_like(Y)*w[1]


    # plot wind vector field
    plt.quiver(X, Y, U, V, angles='xy', scale_units='xy', scale=1,
            alpha = 0.65, color = wind_color, label = f'Wind Velocity = {wind_velocity}

    # plot max range and height vs launch angle
    plt.plot(launch_angles,max_ranges_with_wind, label = 'Max Range')
    plt.plot(launch_angles,max_heights_with_wind, label = 'Max Height')
    plt.xlabel('Launch Angle (deg)')
    plt.ylabel('Distance (m)')
    plt.axhline(y=0, color='black', linestyle='-')
    plt.axvline(x=0, color='black', linestyle='-')
    plt.xticks(np.arange(0, max(launch_angles)+5, step=5))
```

```
plt.yticks(np.arange(0, int(np.ceil((max(max_ranges_with_wind)/5))*5)+5, step=5))
plt.xlim(0, np.ceil(max(launch_angles)/5)*5)
plt.ylim(0, int(np.ceil((max(max_ranges_with_wind)/5))*5))
plt.title(f'Max Range and Height vs Launch Angle launched at {v0} m/s with Wind Vel
plt.legend()
plt.grid()
plt.show()
```

The graphs for all above models are found in section 4.2

3.3: Exercise 3

Recall from section **2.3** eq. (40) is a quadratic equation. $\tan(\theta)$ and $\frac{g\sec^2(\theta)}{2v_0^2}$ can be treated as contants. Thus, we can use the quadratic formula to solve for θ.

let's suppose that diego is 30 meters away from mia. We can use the quadratic formula to solve for θ.

To find the appropriate launch angle so that diego kicks the ball to Mia's head, recall (48):

$$\theta = \arctan\left(\frac{-(-x) \pm \sqrt{(-x)^2 - 4\left(\frac{gx^2}{2v_0^2}\right)\left(y + \frac{gx^2}{2v_0^2}\right)}}{2\left(\frac{gx^2}{2v_0^2}\right)}\right) \tag{49}$$

Plugging in the values gets:

$$\theta = \arctan\left(\frac{-(-30) \pm \sqrt{(-30)^2 - 4\left(\frac{9.81(30^2)}{2(30)^2}\right)\left(2 + \frac{9.81(30^2)}{2(30)^2}\right)}}{2\left(\frac{9.81(30^2)}{2(30)^2}\right)}\right) \tag{50}$$

$$\theta = \arctan\left(\frac{30 \pm \sqrt{900 - 4\left(\frac{9.81}{2}\right)\left(2 + \frac{9.81}{2}\right)}}{9.81}\right) \tag{51}$$

$$\theta = \arctan\left(\frac{30 \pm \sqrt{900 - 2(9.81)\left(2 + \frac{9.81}{2}\right)}}{9.81}\right) \tag{52}$$

$$\theta = \arctan\left(\frac{30 \pm \sqrt{900 - 4(9.81) - 9.81^2}}{9.81}\right) \tag{53}$$

$$\theta = \arctan\left(\frac{30 \pm \sqrt{764.5239}}{9.81}\right) \tag{54}$$

$$\theta = \arctan\left(\frac{30 + \sqrt{764.5239}}{9.81}\right) \tag{55}$$

$$\theta = 80.343^\circ \tag{56}$$

$$\theta = \arctan\left(\frac{30 - \sqrt{764.5239}}{9.81}\right) \tag{57}$$

To get θ in degrees, multiply by $\frac{180}{\pi}$.

$$\theta = \arctan(\frac{30 - \sqrt{764.5239}}{9.81})(\frac{180}{\pi}) \tag{58}$$

$$\theta = 13.471^\circ \tag{59}$$

In this case, both solutions are valid, so if Diego were 30 meters away from Mia and kicked the ball at 30 $\frac{m}{s}$ he could kick it either with a launch angle of 80.343 degrees or 13.471 degrees.

It is important to note that the quadratic formula can have 0, 1, or 2 solutions. If the discriminant is negative, there are no real solutions. If the discriminant is 0, there is one real solution. If the discriminant is positive, there are two real solutions.

The above equations are represented in the following function.

```
def find_launch_angle(x,v0,y = 2):
    """x = relative distance, v0 = initial velocity, y  = height of Mia's head"""
    pm = np.array([+1, -1])
    a = (g*x**2)/(2*v0**2)
    b = -x
    c = (y+(((x**2)*g)/(2*v0**2)))
    solutions = np.rad2deg(np.arctan((-b + (pm * np.sqrt((b**2)-(4*a*c))))/(2*a)))
    # if no soultions:
    if b**2-4*a*c < 0:
        return 'no solutions, try a higher velocity'
    # if one solution:
    elif b**2-4*a*c == 0:
        solution = solutions[~np.isnan(solutions)] # get the non-nan value
        t = x/(v0*np.cos(np.deg2rad(solution))) # time it takes to get to Mia
        vy = v0*np.sin(np.deg2rad(solution)) - g*t # y velocity at time it takes to get
        if vy>=0: # if y velocity is greater than 0, then the ball reached Mia, but not
            return 'ball reach Mia,but not while it was falling downward'
        else:
            return solution
    # if two solutions:

    # find solutions where ball is v_y of ball is less than 0
    for solution in solutions:
        t = x/(v0*np.cos(np.deg2rad(solution)))
        vy = v0*np.sin(np.deg2rad(solution)) - g*t
        if vy>=0:
            solutions = solutions[solutions!=solution]
    if len(solutions) == 0:
        return 'ball reached Mia, but not while it was falling downward'
    return solutions
```

The function above takes in the distance between Diego and Mia, the initial velocity of the soccer ball, and the height of Mia's head. The function either returns the launch angle(s) that will get the soccer ball to Mia's head on it's descent, or if the function returns a string, it means that the soccer ball will not reach Mia's head. Or that it did, but not on it's descent.

The following code plots the angles to check if they are valid, or prints the string indicating the issue if they are not valid

```
distances = [10,20,30,40,50,60,100]
for distance in distances:
    thetas = find_launch_angle(distance,30)
    #if theta is a string, then print there are no solutions
    if isinstance(thetas,str):
        print(thetas+f'. There are no solutions for distance = {distance}')
        continue
    count=0
    for theta in thetas:
        count+=1
        x_graph, y_graph= soccer_ball_trajectory_no_drag(0,0,30,theta,ball_and_environm
        plt.plot(x_graph,y_graph, label = f'Diego to Mia, $\\theta_{count}$ = {theta:.2
    plt.scatter(distance,2, label = 'Mia', marker='o', color='red')
    plt.scatter(0,0, label = 'Diego', marker='o', color='blue')
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
    plt.title('Diego to Mia')
    plt.xlim(0, max(50,distance+5))
    plt.ylim(0,max(50,max(y_graph)+5))
    plt.xticks(np.arange(0, max(50,distance+5), step=5))
    plt.yticks(np.arange(0,max(50,max(y_graph)+5) , step=5))
    plt.legend()
    plt.axhline(y=0, color='black', linestyle='-')
    plt.axvline(x=0, color='black', linestyle='-')
    plt.show()
```

EDIT:

As mentioned in **2.3**, I realized that I was not accounting for the air resistance when that is the purpose of the project. Given the time constraints I am working under, I have implemented a coded solution that I believe gives the correct angle given the conditions, however it is found through brute force and a little estimation, and not actual analytical methods. The code is found below and the results are found in section 4.3.

```
def soccer_ball_trajectory_new(
       x0 = 0,
       y0 = 0,
```

```
    target_x = 30,
    target_y = 2,
    v0 = 30,
    launch_angle = 30,
    ball_and_environment_properties= ball_and_environment_properties,
    wind_velocity = 0,
    wind_angle = 0,
    tolerance = 0.01):
# Soccer ball and environment constants
diameter = ball_and_environment_properties['diameter']
mass = ball_and_environment_properties['mass']
density_air = ball_and_environment_properties['density_air']
Cd = ball_and_environment_properties['Cd']
g = ball_and_environment_properties['g']
A = np.pi*(diameter/2)**2   # Cross-sectional area of soccer ball

# Initial Conditions
nstep = 100000
t = np.linspace(0., 60., nstep)
alpha = 0.5*Cd*A*density_air
w = wind(wind_velocity,wind_angle)
v = velocity(v0,launch_angle)
v_magnitude = velocity_magnitude(v)
unit_v = velocity_unit_vector(v,v_magnitude)
a = acceleration(g)
F = force(a,mass)
F_drag = drag_force(alpha,v_magnitude,unit_v)
F_net = net_force(F,F_drag)
x = x0
y = y0
x_graph = np.zeros_like(t)
y_graph = np.zeros_like(t)
x_graph[0] = x
y_graph[0] = y
i = 0
while y_graph[i] >= 0 and x_graph[i] <= target_x:
    rel_v = [v[0]-w[0], v[1]-w[1]]
    v_magnitude = np.sqrt(rel_v[0]**2+rel_v[1]**2)
    unit_v = rel_v/v_magnitude
    F_drag = alpha*(v_magnitude**2)*-unit_v
    F = [a[0]*mass, a[1]*mass]
    F_net = [F[0]+F_drag[0], F[1]+F_drag[1]]
    v = [v[0]+(F_net[0]/mass)*dt, v[1]+(F_net[1]/mass)*dt]
    x = x + v[0]*dt
    y = y + v[1]*dt
    if x >= target_x:
        x_graph[i+1] = x
        y_graph[i+1] = y
        break
```

```
        if y == 2:
            y_graph[i+1] = y
            break
        if y<=y_graph[i-1] and y < target_y:
            y_graph[i+1] = y
            break
        x_graph[i+1] = x
        y_graph[i+1] = y
        i+=1
    x_graph = x_graph[:i]
    y_graph = y_graph[:i]
    if len(x_graph) == 0:
        msg = str("miss pass")
        return msg, x_graph,y_graph,x0,y0,target_x,target_y ,v0,launch_angle,ball_and_en
    if abs(x_graph[-1]-target_x) > tolerance:
        msg = str("miss pass")
        return msg, x_graph,y_graph,x0,y0,target_x,target_y ,v0,launch_angle,ball_and_en
    elif abs(y_graph[-1]-target_y) > tolerance:
        msg = str("miss pass")
        return msg, x_graph,y_graph,x0,y0,target_x,target_y ,v0,launch_angle,ball_and_en
    return x_graph,y_graph,x0,y0,target_x,target_y ,v0,launch_angle,ball_and_environment

def new_find_launch_angle(x0 = 0,
                          y0 = 0,
                          target_x = 30,
                          target_y = 2,
                          v0=30,
                          ball_and_environment_properties= ball_and_environment_properti
                          wind_velocity=0,
                          wind_angle=0,
                          tolerance = 0.1):

    # Iterate over a range of launch angles (e.g., from 10 to 80 degrees)
    valid_angles = []
    angles = np.linspace(0,90,int(91/tolerance))
    plt.figure(figsize=(10,10))

    for angle in angles:
        results = soccer_ball_trajectory_new(x0 = x0, y0 = y0,target_x=target_x,target_y
        #if results[0] is a string
        if isinstance(results[0],str):
            x_graph, y_graph = results[1], results[2]
            # print(results[0])
        else:
            x_graph, y_graph, launch_angle = results[0], results[1], results [7]
            if y_graph[-1]>y_graph[-2]:
                print(f"The ball launched at {v0} m/s at {angle}º will reach Mia's head,
                continue
            else:
```

```
                # if there is an angle in valid_angles that is within 0.1 deg of the cur
                if len(valid_angles)>0:
                    if any(abs(angle-valid_angles)<tolerance):
                        continue
                valid_angles.append(angle)
                w = wind(wind_velocity,wind_angle)
                x_ceiling = int(np.ceil(x_graph[-1]/5)*5)
                y_ceiling = int(np.ceil(y_graph[-1]/5)*5)
                x_floor = int(np.floor(x_graph[-1]/5)*5)
                y_floor = int(np.floor(y_graph[-1]/5)*5)
                # create grid
                X = np.linspace(min(0,x_floor), max(50,x_ceiling), 5)
                Y = np.linspace(min(0,y_floor), max(40,y_ceiling), 5)
                Y, X = np.meshgrid(Y, X)
                U = np.ones_like(X)*w[0]
                V = np.ones_like(Y)*w[1]
                # plot wind vector field
                if wind_velocity == 0:
                    wind_color = 'black'
                elif wind_velocity > 0:
                    wind_color = 'blue'
                else:
                    wind_color = 'red'

                plt.quiver(X, Y, U, V, angles='xy', scale_units='xy', scale=1,
                alpha = 0.65, color = wind_color, label = f'Wind Velocity = {wind_velocity
                plt.plot(x_graph, y_graph, label=f'Launch angle = {launch_angle:.2f} deg
    plt.scatter(target_x, target_y, label='Mia', marker='o', color='red')
    plt.scatter(0, 0, label='Diego', marker='o', color='blue')
    plt.xlabel('x (m)')
    plt.ylabel('y (m)')
    plt.title('Diego to Mia at initial velocity = {v0} m/s')
    plt.xlim(0, max(50, target_x + 5))
    plt.ylim(0, max(50, max(y_graph) + 5))
    plt.xticks(np.arange(0, max(50, target_x + 5), step=5))
    plt.yticks(np.arange(0, max(50, max(y_graph) + 5), step=5))
    plt.legend()
    plt.axhline(y=0, color='black', linestyle='-')
    plt.axvline(x=0, color='black', linestyle='-')
    plt.show()

    print(f"there are roughly {valid_angles} valid angles that hit Mia's head descending

    return
```

The code above attempts to basically simulate many angles until they are within a certain tolerance of the target distance and height for Mia's head. It is computationally intensive, and I am sure there is a better way to do it, but it works. The time to compute increases as the tolerance decreases. So

the more accurate results desired, the less long it will take to calculate. The results are found in section 4.3.

A quick run through of the code:

`soccer_ball_trajectory_new()` is adjusted to from the regular socer_ball_trajectory() method to filter out values trajectories that are not within a certain tolerance level of the target: Mia's head

`new_find_launch_angle()`

is the function that iterates over a range of launch angles from 0 to 90 degrees and calls `soccer_ball_trajectory_new()` to get the trajectory of the soccer ball for each angle. If the trajectory is valid, it is plotted. If not, it is not plotted. The function also keeps track of the valid angles and prints them out at the end.

3.4: Exercise 4

For this question, I simply repurposed the code from **3.3** to find the approximate launch angles for every velocity starting from 0 and rising until the maximum velocity at $58.6\frac{m}{s}$. Once I found a the first valid launch angle for the slowest possible velocity, I stopped the loop. Below is the code to find the launch angle and plot the trajectory:

```
tolerance = 1.22
wind_velocity = 0
wind_angle = 0

velocities  = np.linspace(0,58.6,int(58.6/tolerance))

for v0 in velocities:
    valid_angles,x_graphs,y_graphs = new_find_launch_angle(x0 = 0, y0 = 0 ,target_x = 1
    if len(valid_angles) == 0:
        print(f'v0 = {v0} m/s, no solutions')
        continue
    elif len(valid_angles) >= 1:
        valid_angles = valid_angles[:1]
        x_graphs = x_graphs[:1]
        y_graphs = y_graphs[:1]

        # print(f'hello world: \t {x_graphs[0][-1]}')
        #plot_new(x_graphs,y_graphs,valid_angles,11,0,0,0,xlim = 11,ylim=20,ticks = 1)
        print(f'A good spoon shot from this distance given the conditions would be {v0}
        print(f'x_graphs = {x_graphs}, y_graphs = {y_graphs}')
        break
```

```
plot_new(x_graphs,y_graphs,valid_angles,11,0,0,0,xlim = 15,ylim=5,ticks = 1)
```

I made one small change to the plotting code which is just changingbthe label from Mia to Goal Line. To not be redundant I will not show the code here.

4. Results and Discussion

The following subsections will contain the the results and analysis of of the results from the above exercises.

4.1: Exercise 1

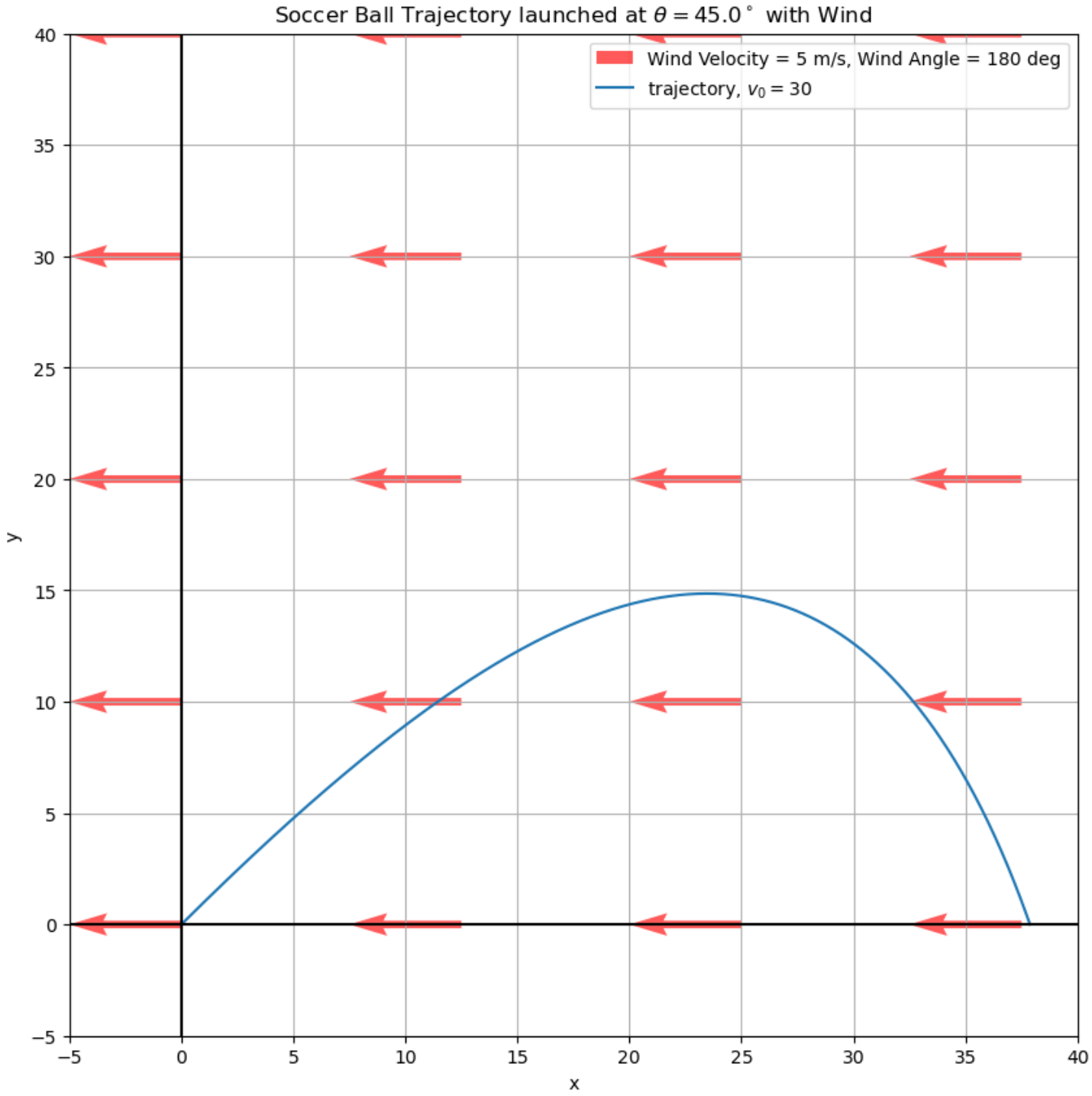


Figure 4.1.1: Trajectory of a soccer ball launched at 45° with an initial velocity of 30 $\frac{m}{s}$ with $5\frac{m}{s}$ of horizontal wind at 180° (or $-5\frac{m}{s}$ of wind at 0°) and no vertical wind.

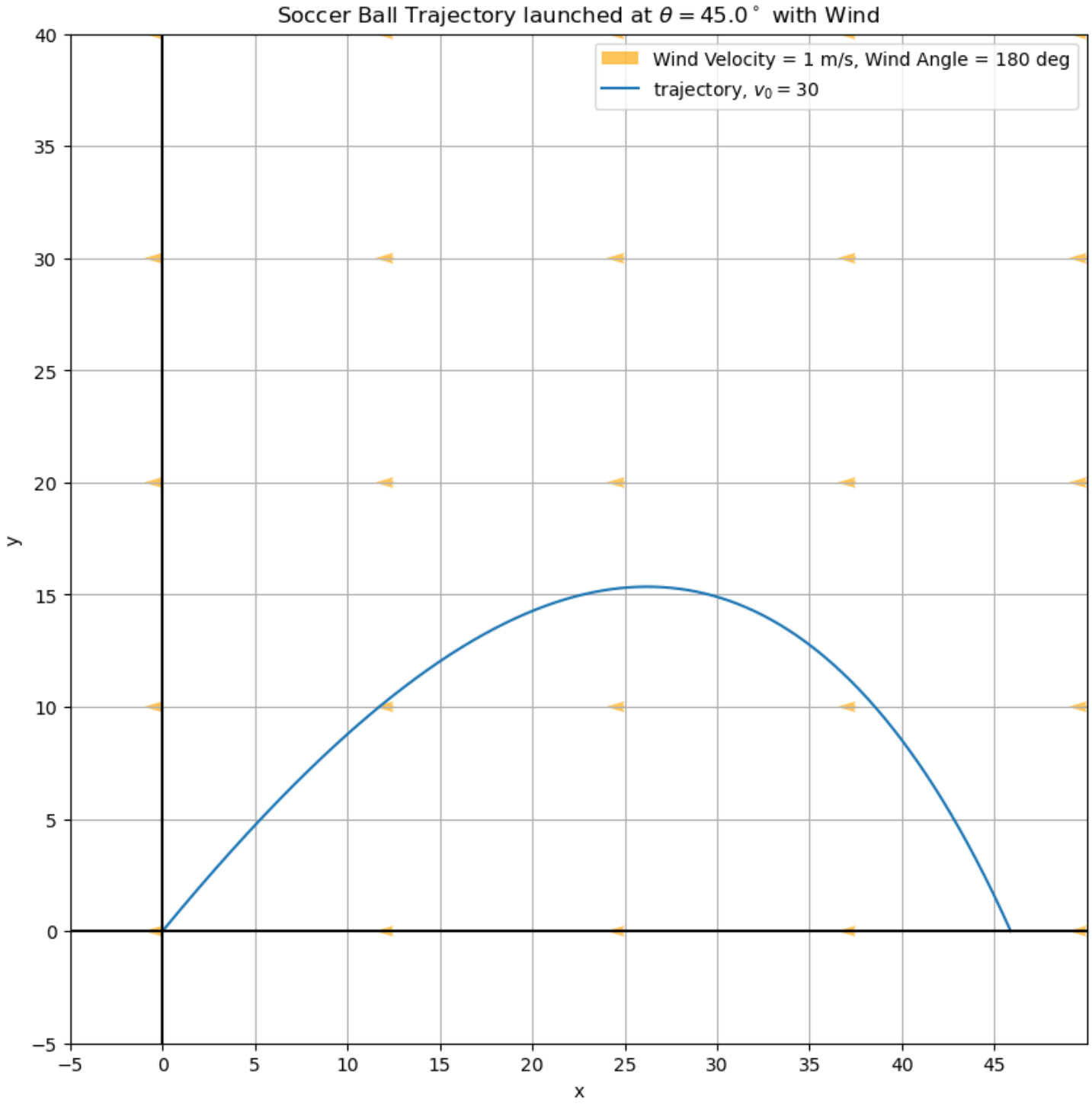


Figure 4.1.2: Trajectory of a soccer ball launched at 45° with an initial velocity of $30\frac{m}{s}$ with $1\frac{m}{s}$ of wind at 180° (or $-1\frac{m}{s}$ of wind at 0°)

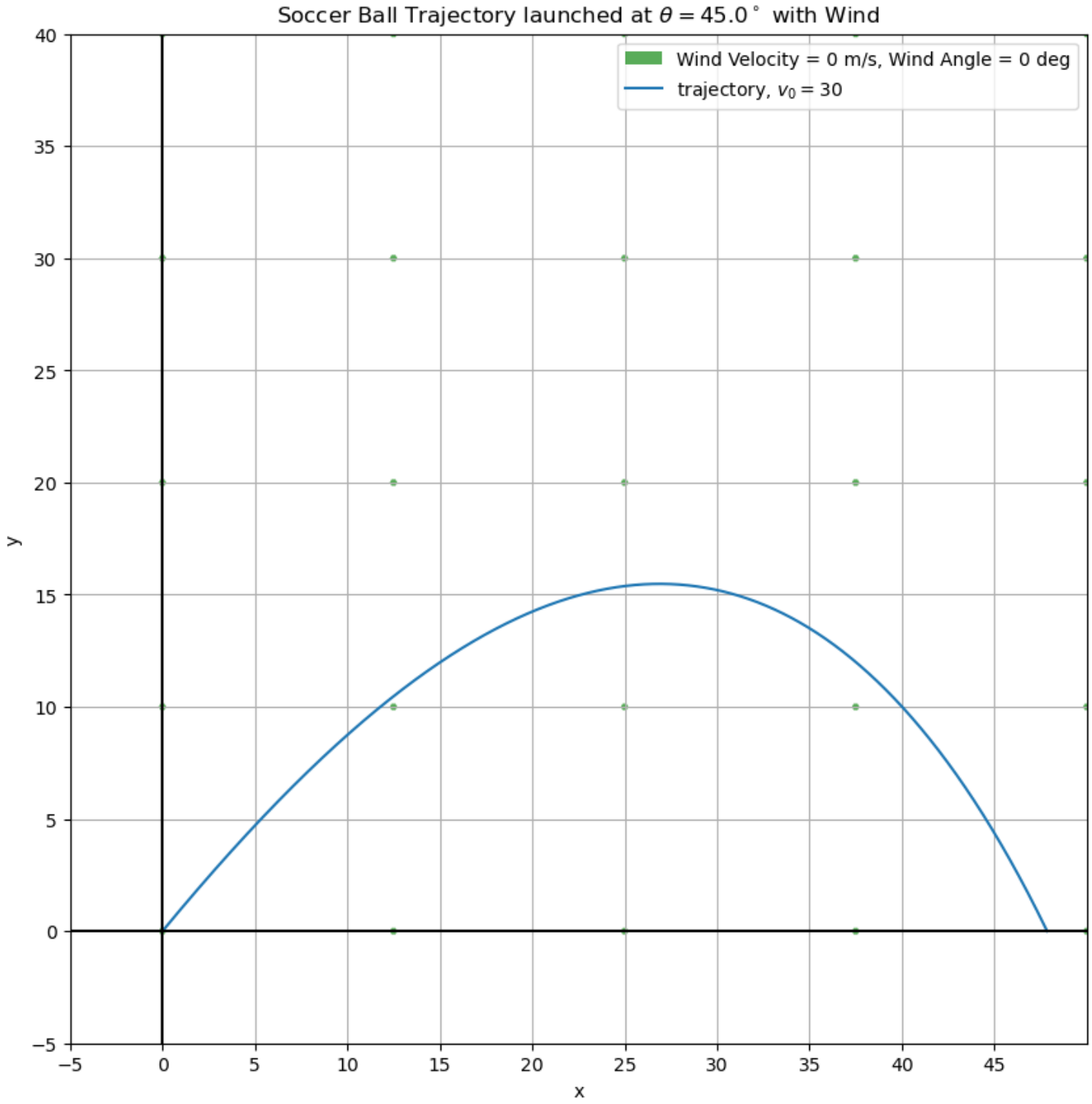


Figure 4.1.3: Trajectory of a soccer ball launched at 45° with an initial velocity of 30 $\frac{m}{s}$ with no wind.

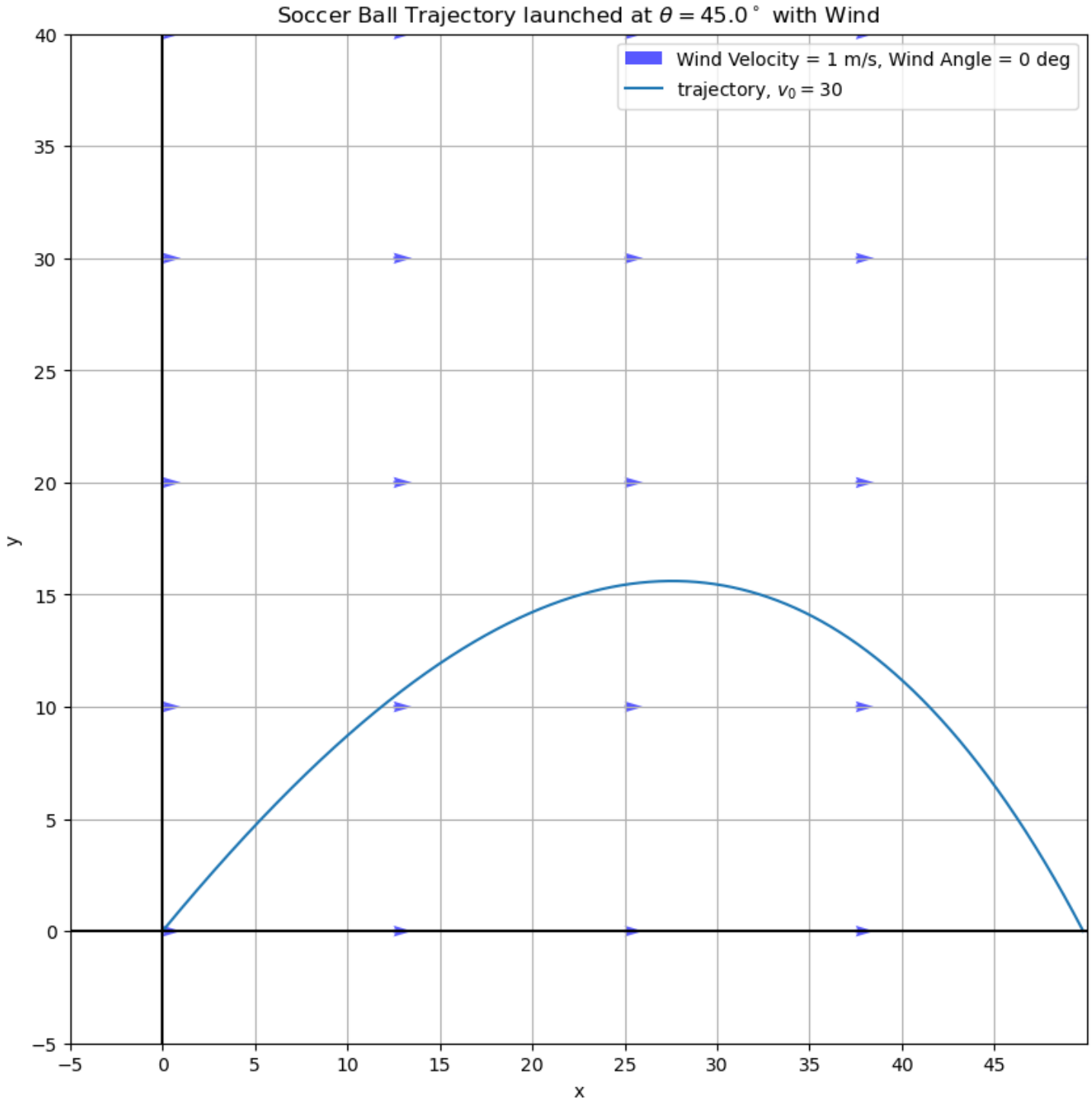


Figure 4.1.4: Trajectory of a soccer ball launched at 45° with an initial velocity of 30 $\frac{m}{s}$ with 1 $\frac{m}{s}$ of wind at 0°.

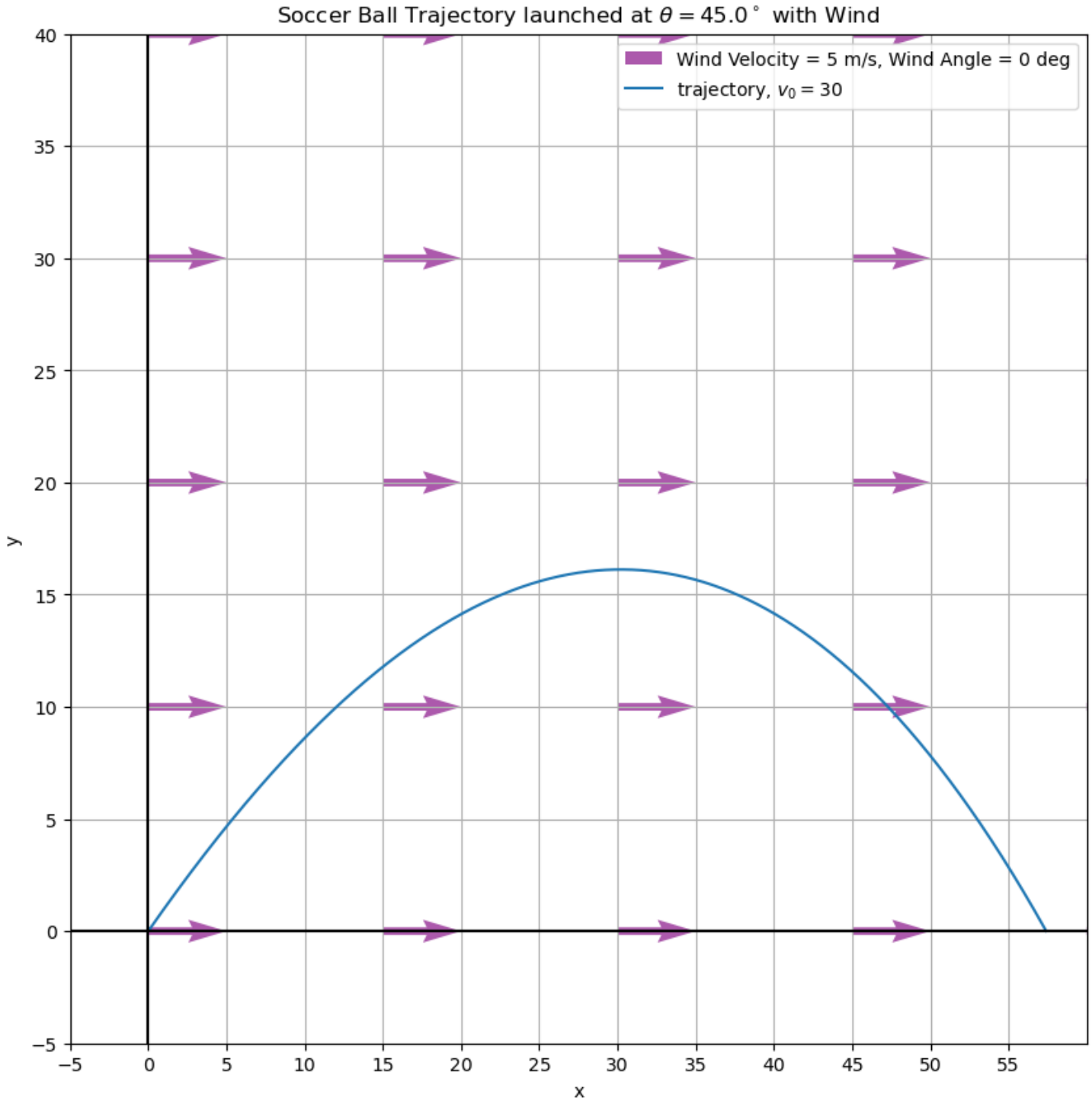


Figure 4.1.5: Trajectory of a soccer ball launched at 45° with an initial velocity of 30 $\frac{m}{s}$ with 5 $\frac{m}{s}$ of wind at 0°.

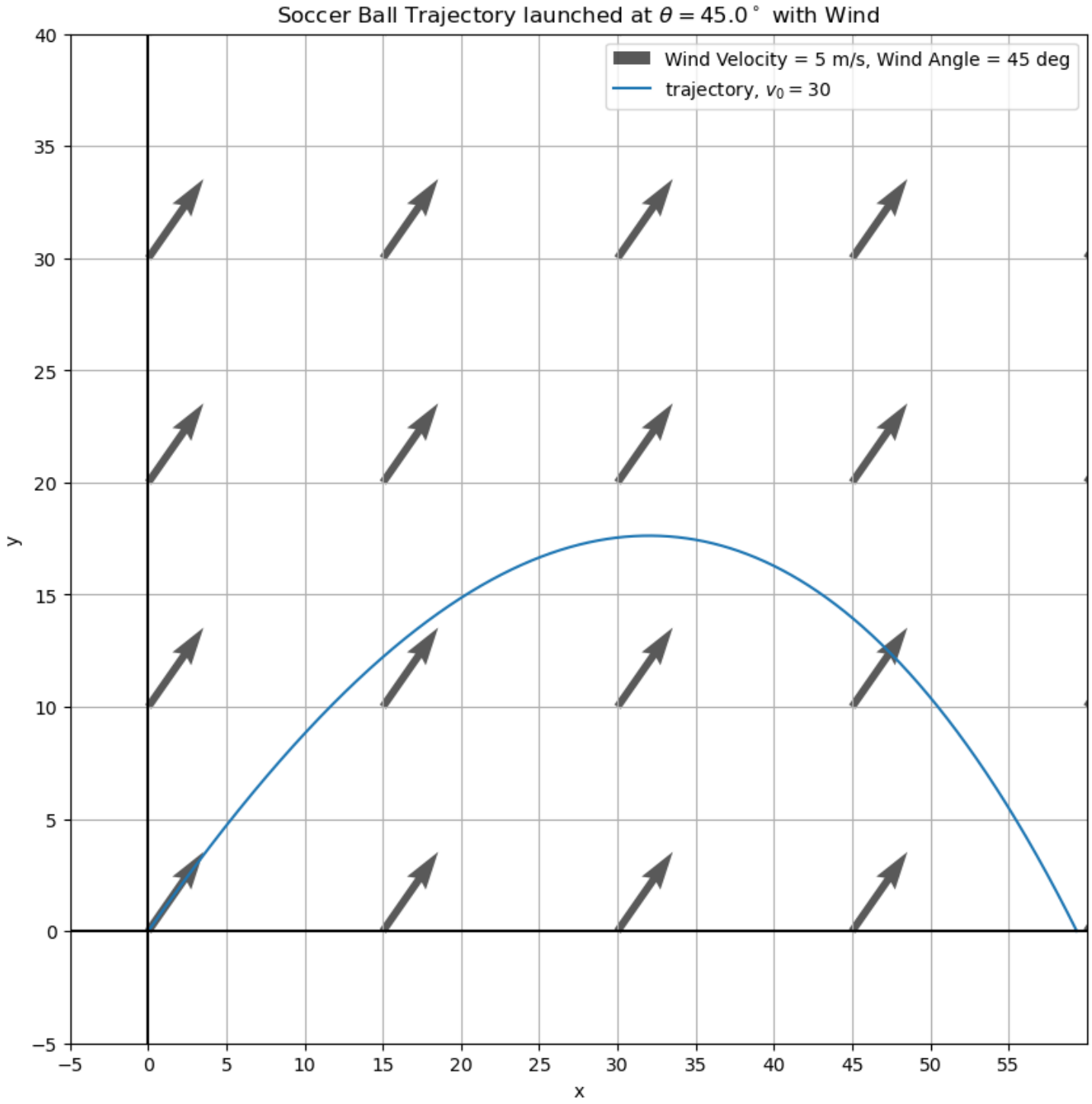
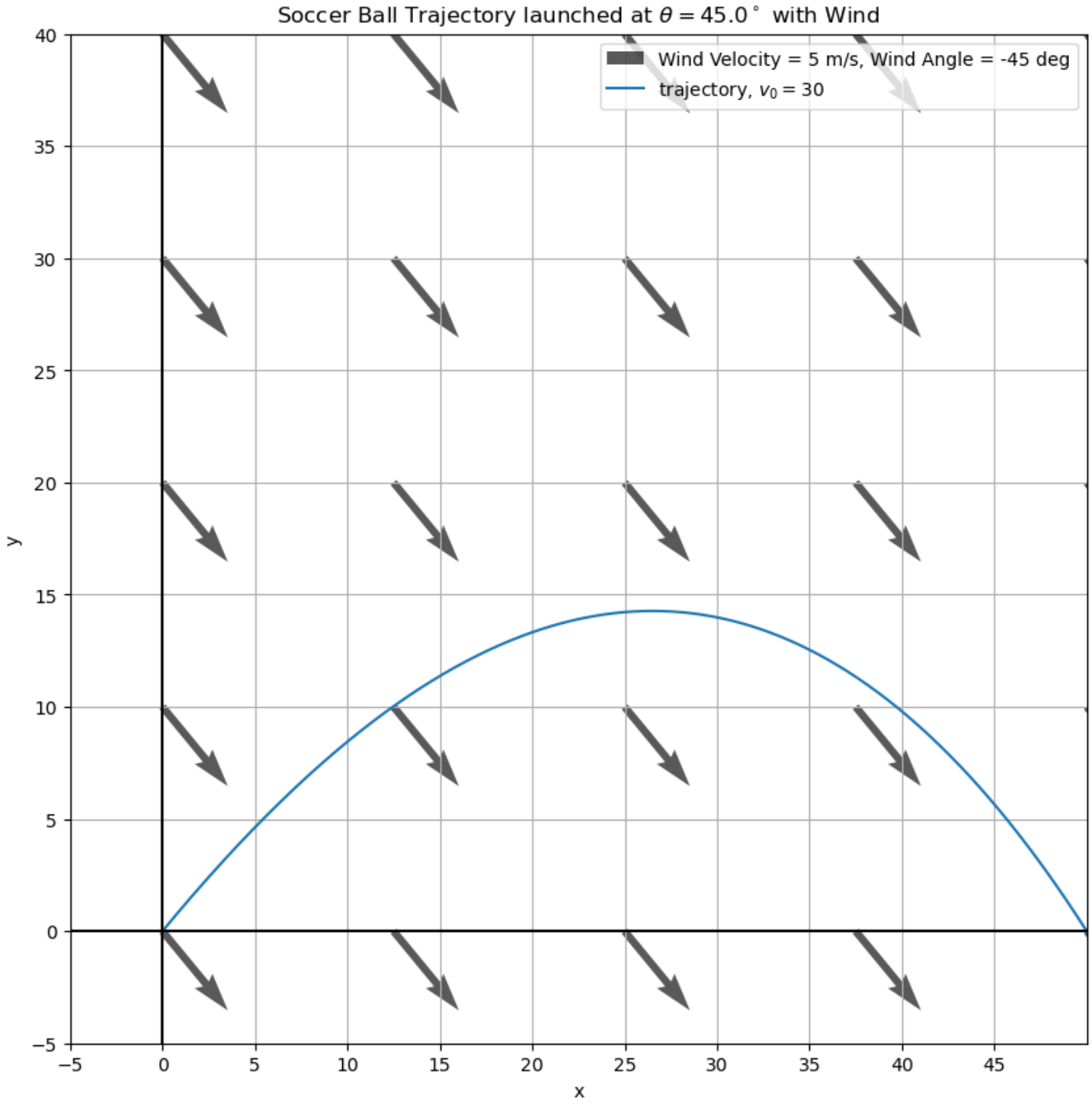


Figure 4.1.6: Trajectory of a soccer ball launched at 45° with an initial velocity of $30\frac{m}{s}$ and $5\frac{m}{s}$ of wind at 45°



> Figure 4.1.7: Trajectory of a soccer ball launched at 45° with an initial velocity of $30\frac{m}{s}$ and $5\frac{m}{s}$ of wind at -45°

The graphs above demonstrate the hypothesis expected of a projectile that experiences drag in different wind environments.

For all graphs above, as v_{wind_x} increases, the range increases. Additionally, as v_{wind_x} decreases, the range decreases.

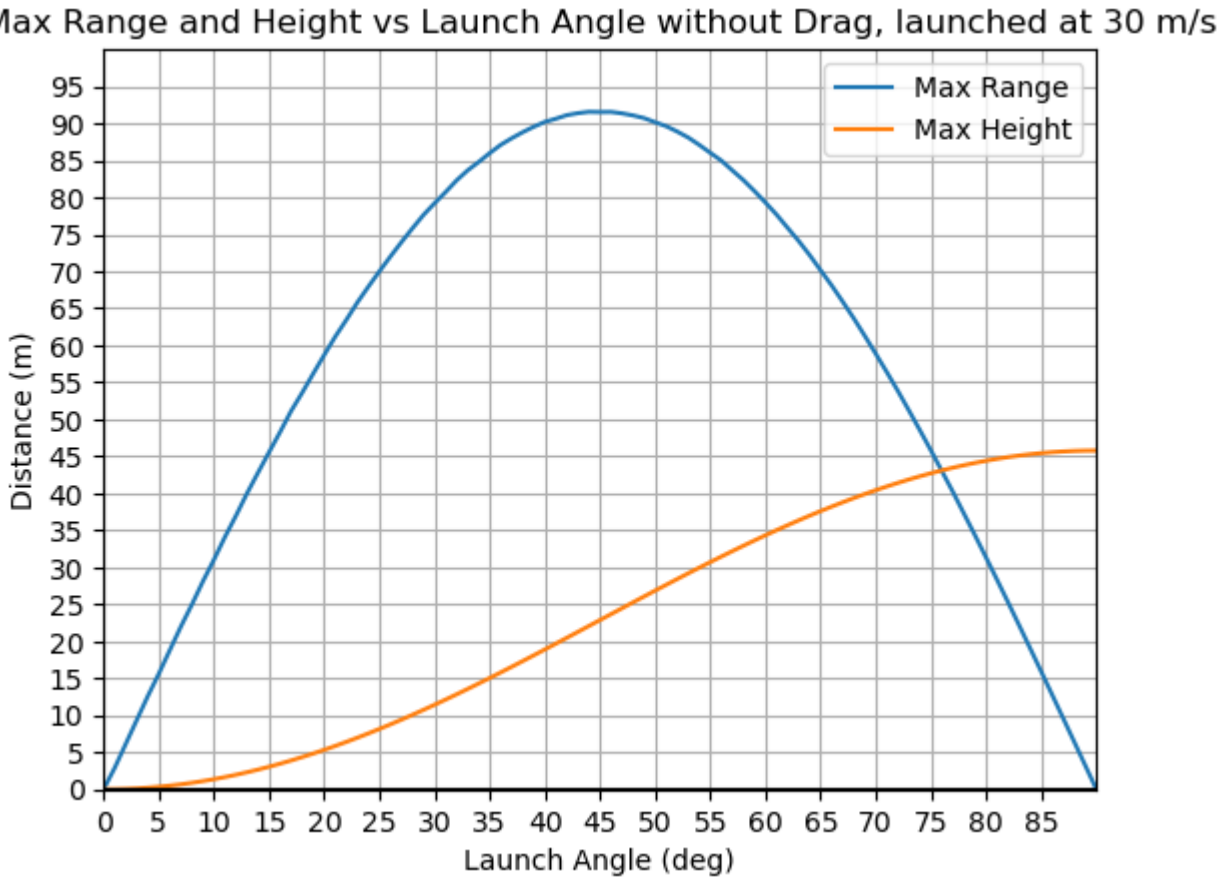
Similarly, as v_{wind_y} increases, the height increases. Additionally, as v_{wind_y} decreases, the height decreases.

These results sense because intuitively a tailwind (wind travelling in the direction of the ball's motion) will push the ball forward, increasing the range of the soccer ball. Similarly a a headwind (wind travelling in the opposite direction of the ball) will push against the ball, decreasing the range of the soccer ball. Likewise, An updraft (upward wind) will push the ball up, increasing the height of the soccer ball, and a downdraft (downward wind) will push the ball down, decreasing the height of the soccer ball.

4.2: Exercise 2

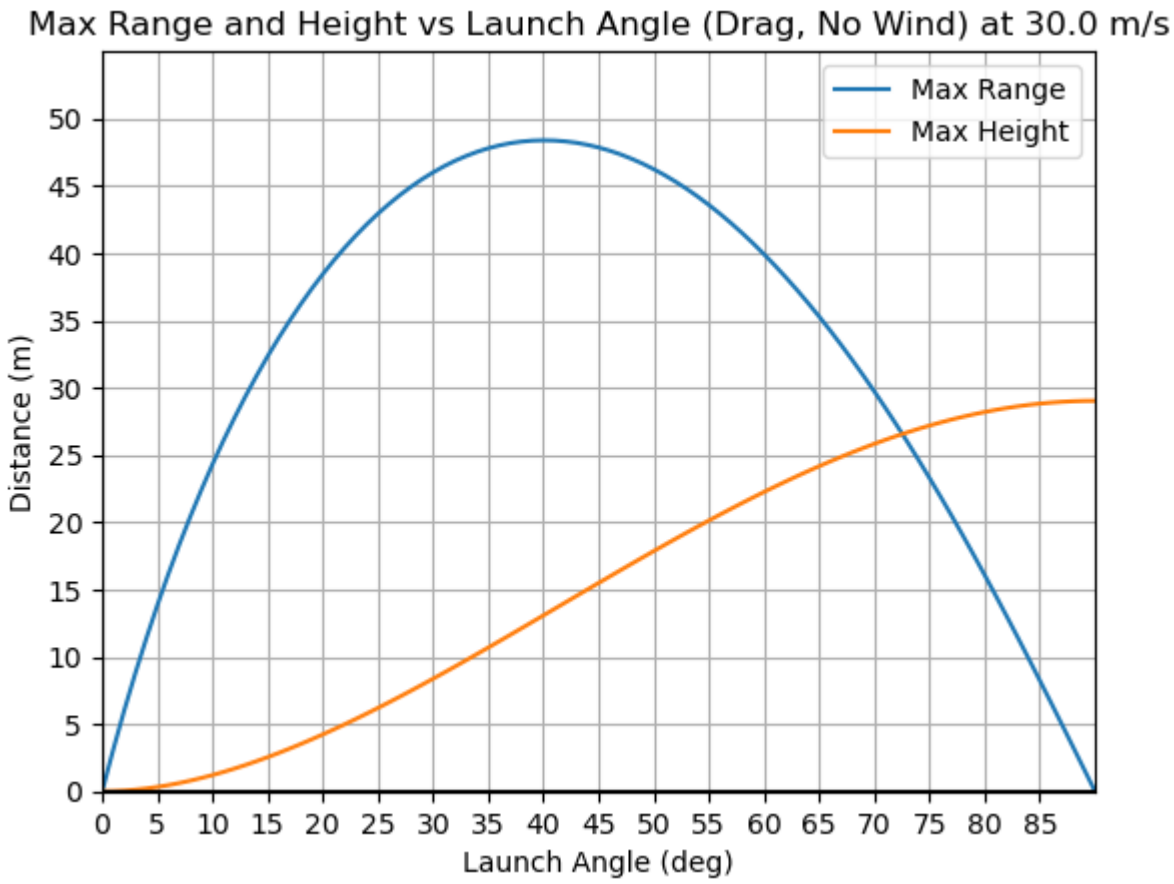
Section **3.2** demonstrated the implementation of various models of the trajectory of a soccer ball under the same launch conditions within different environments.

The first environment had no drag. The graph below shows the maximum height and range of a soccer ball launched at a given angle with the given average professional kick velocity, $30\frac{m}{s}$ with no drag and no wind.



> Figure 4.2.1: Maximum range and height for launch angles between 0 and 90 degrees with no drag and no wind.

The second environment had drag and no wind. The graph below shows the maximum height and range of a soccer ball launched at a given angle with the given average professional kick velocity, $30\frac{m}{s}$ with drag and no wind.



> Figure 4.2.2: Maximum range and height for launch angles between 0 and 90 degrees with drag and no wind.

The third environment had drag and wind. The following graphs below shows the maximum height and range of a soccer ball launched at a given angle with the given average professional kick velocity, $30\frac{m}{s}$ with drag and winds at various directions and magnitudes.

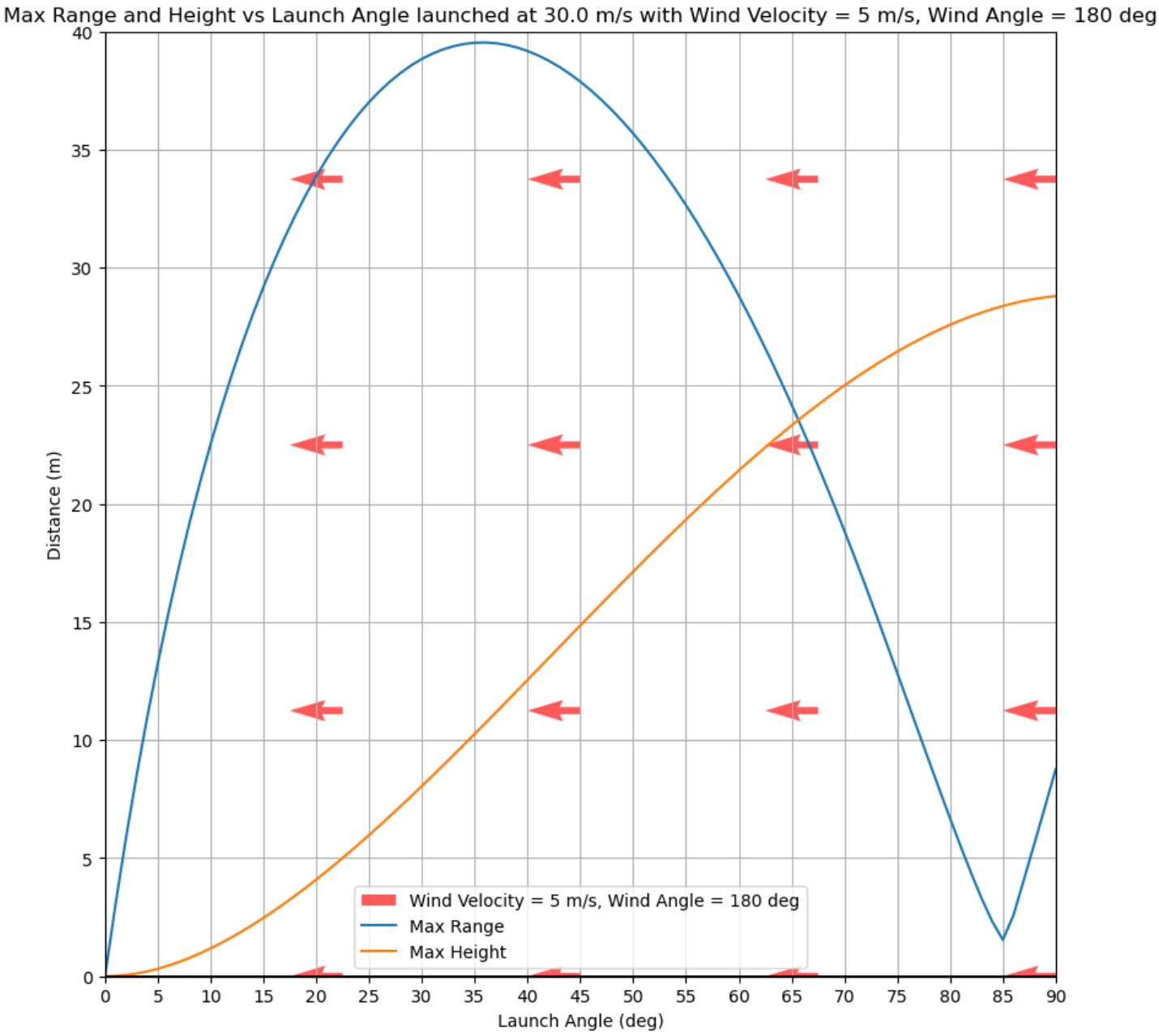
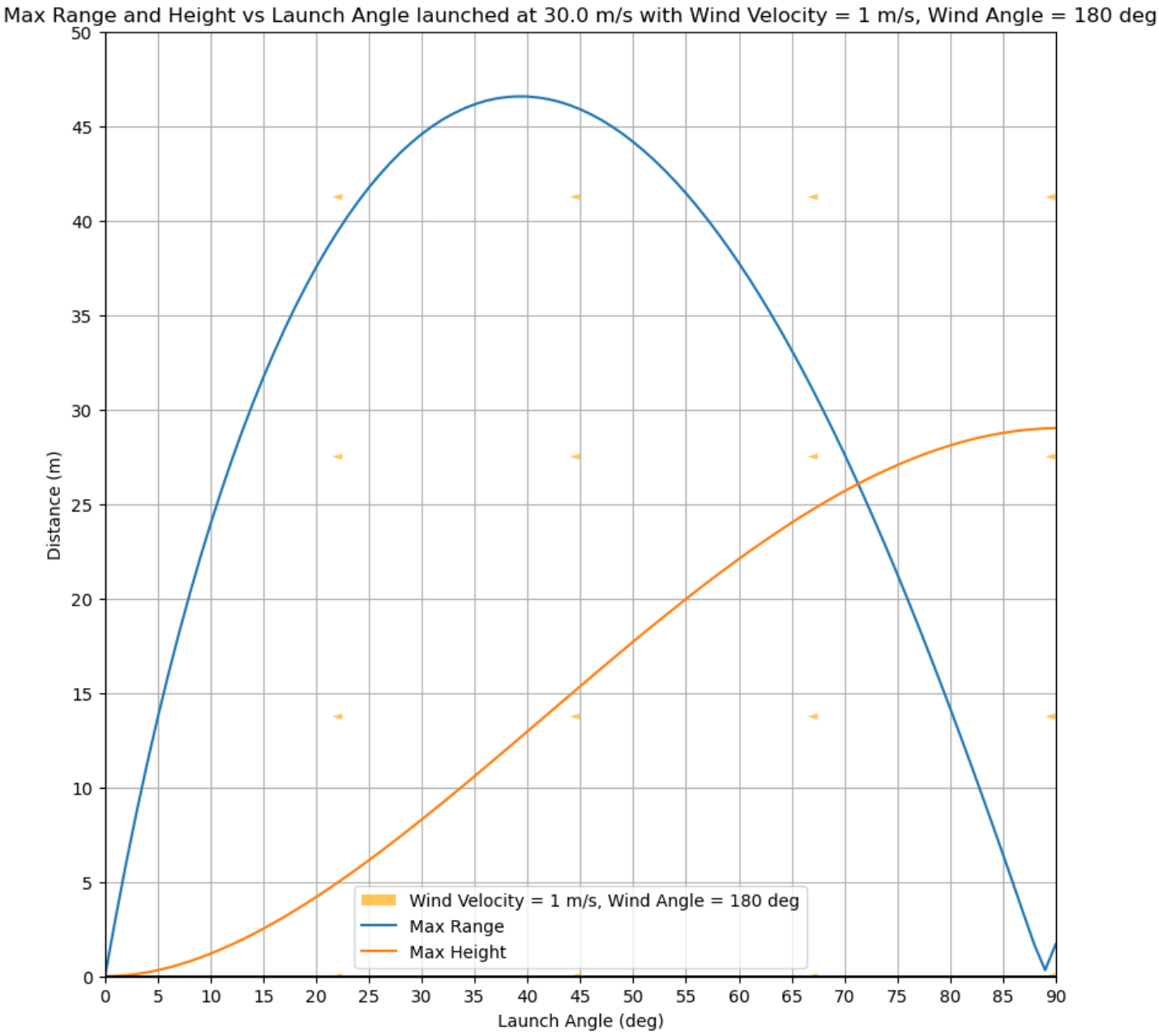


Figure 4.2.3: Maximum range and height for launch angles between 0 and 90 degrees with drag and 5 $\frac{m}{s}$ of wind at 180° to the ball.



> Figure 4.2.4: Maximum range and height for launch angles between 0 and 90 degrees with drag and 1 $\frac{m}{s}$ of wind at 180° to the ball.

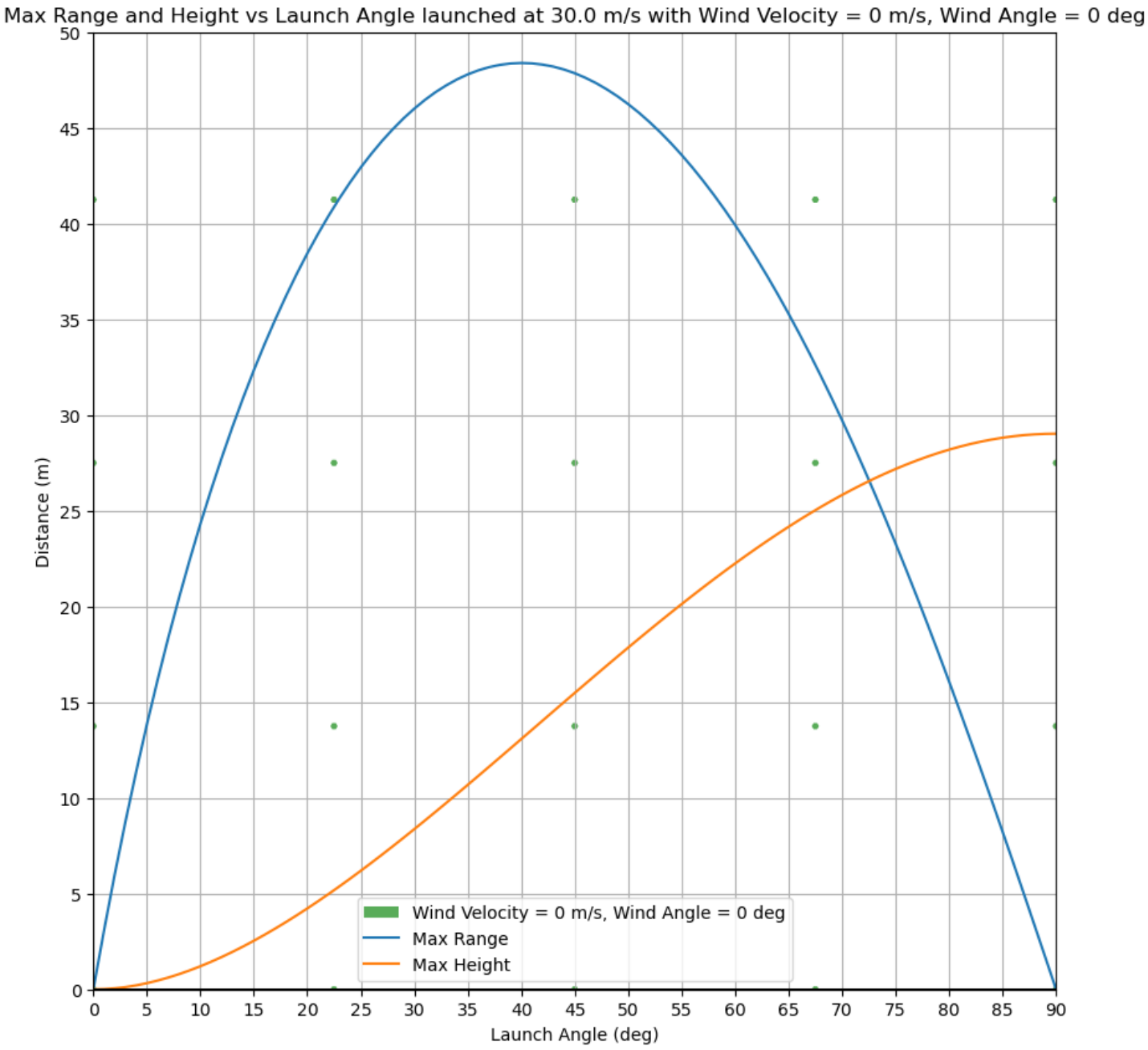


Figure 4.2.5: Maximum range and height for launch angles between 0 and 90 degrees with no drag and wind with velocity 0$\frac{m}{s}$. (notice this is the same as the graph in figure 1, just on a different scale.)

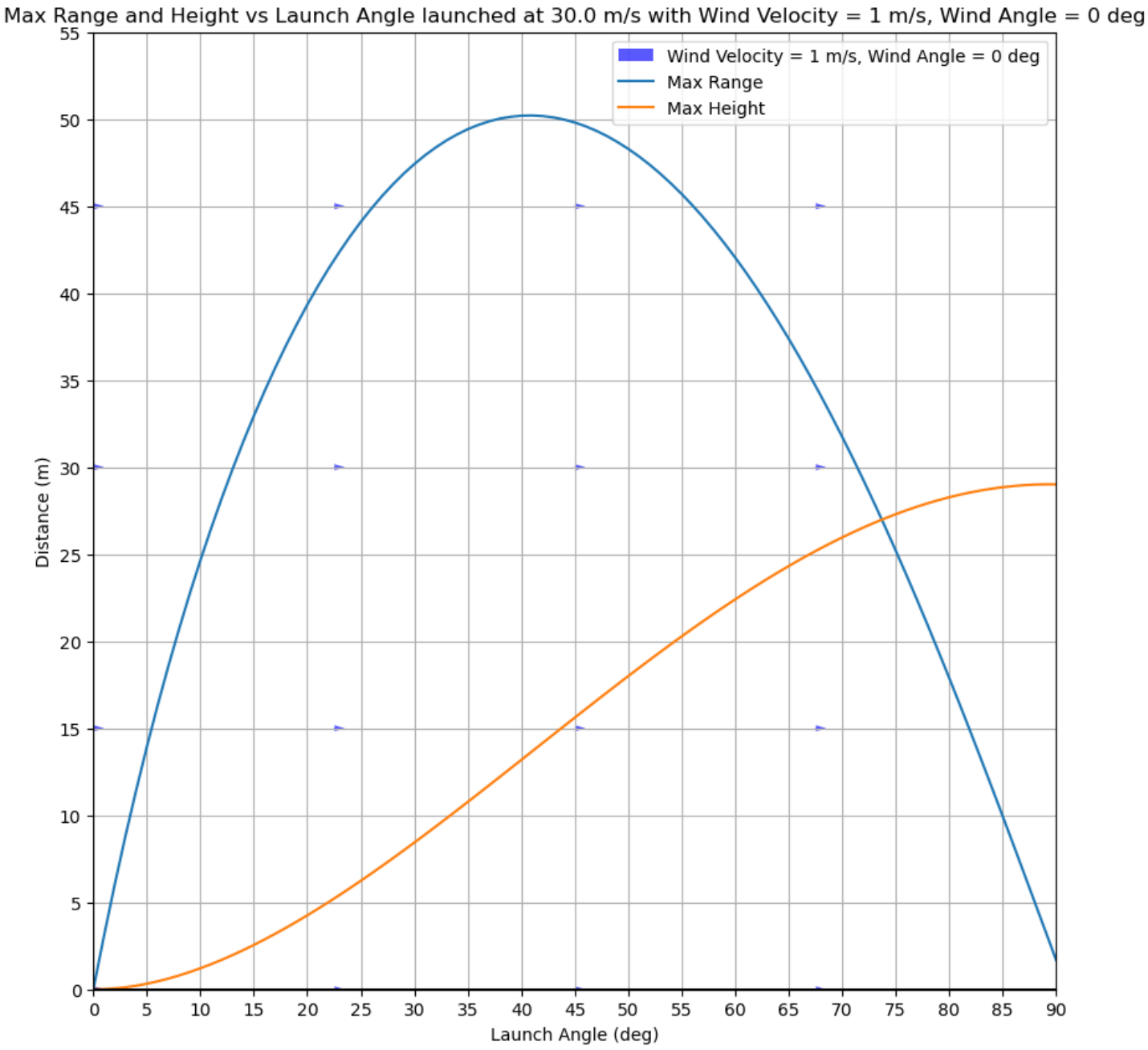
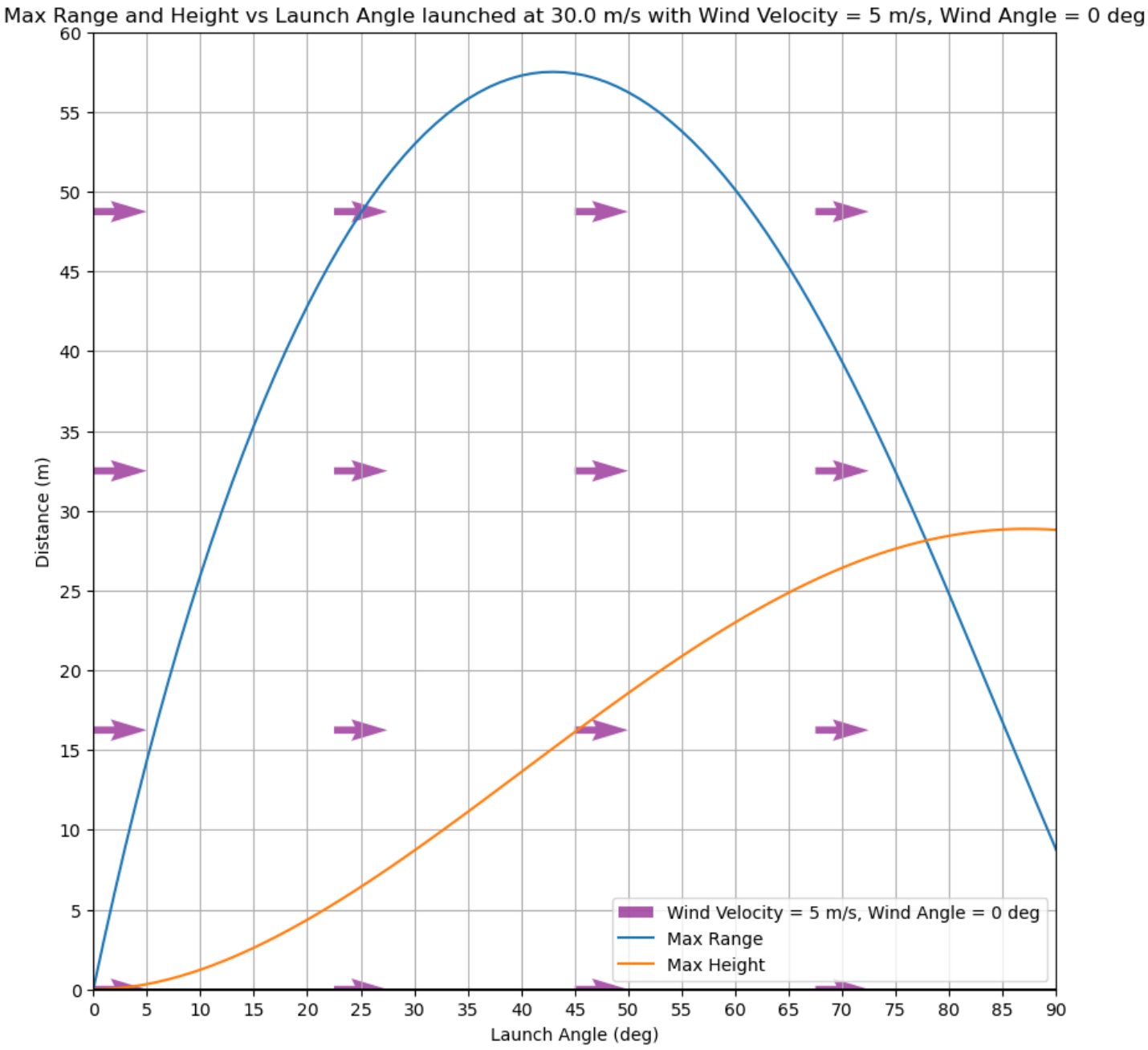


Figure 4.2.6: Maximum range and height for launch angles between 0 and 90 degrees with drag and 1 $\frac{m}{s}$ of wind at 0° to the ball.



> Figure 4.2.7: Maximum range and height for launch angles between 0 and 90 degrees with drag and 5 $\frac{m}{s}$ of wind at 0° to the ball.

Similar to the results discussed in **4.1**, a tailwind will increase the range and a headwind will decrease the range for a ball at any given angle. There is an exception to that headwind rule though, because as Figure **4.2.1** and **4.2.2** demonstrate, a headwind will eventually begin to increase the range of a projectile. Why is that? The following graphs will demonstrate the trajectories of soccer balls that exhibit the same behaviour as Figure **4.2.1** and **4.2.2** indicate:

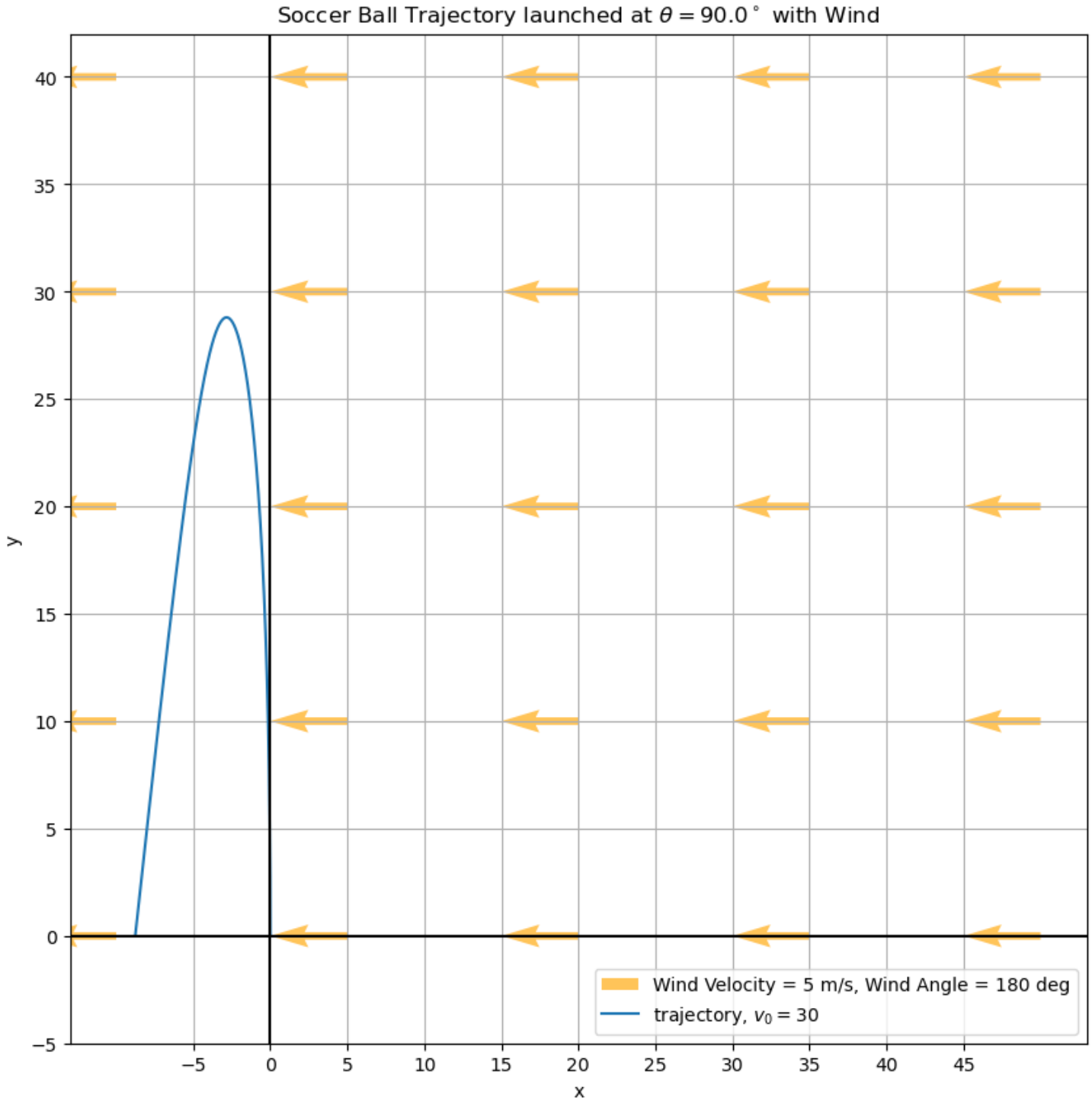
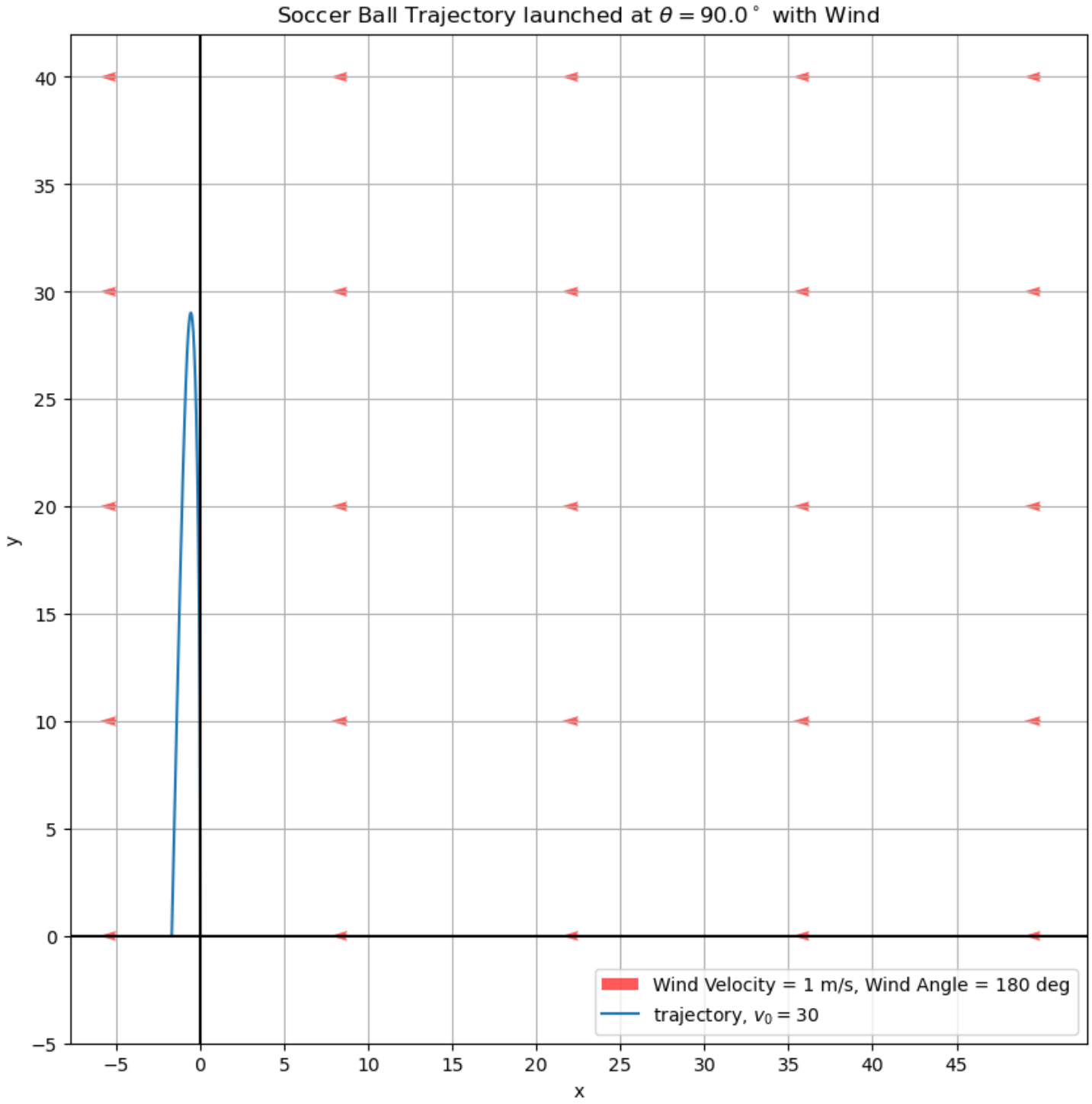


Figure 4.2.8: Trajectory of a soccer ball launched at 90° with an initial velocity of $30\frac{m}{s}$ with 5 $\frac{m}{s}$ of wind at 180° (or $-1\frac{m}{s}$ of wind at 0°)



> Figure 4.2.9: Trajectory of a soccer ball launched at 90° with an initial velocity of $30\frac{m}{s}$ with 1 $\frac{m}{s}$ of wind at 180° (or $-1\frac{m}{s}$ of wind at 0°)

Figures **4.2.8** and **4.2.9** demonstrate that with a strong enough headwind, when the range begins to decrease as the launch angle increases from 0° to 90° the range will approach zero and eventually begin to increase again. This is because the as the angle increases, the v_{0_x} value decreases so much that a at a certain angle the headwind is strong enough that it will cause the relative velocity to be zero, meaning the ball will not travel horizontally and it will land back on the origin.

In the case of the ball in **4.2.1**, this angle is roughly $85°$. In the case of the ball in **4.2.2**, this angle is roughly $89°$.

If the angle increases further, the headwind can push the ball behind the origin. At this point the headwind essentially acts like a tailwind because now the ball is travelling in the direction of the wind. Because the ball is gaining distance from the origin, the range begins to increase again, except this time the ball travels in the negative x direction. Because distance is a scalar value, the range is still increasing, even though the ball is travelling in the negative x direction.

4.2.8 and **4.2.9** demonstrate the backwards motion of the ball when the launch angle is steep enough and the headwind is strong enough to push the ball behind the origin.

A similar phenomenon is seen in figures **4.2.6** and **4.2.7**. In these graphs, the ball is launched at $90°$, so the $v_{0_x} = 0$. However, in both models the headwind is strong enough to push the ball away from the origin in the positive direction, increasing the range. This is why the range does not equal 0 when the ball is launched at $90°$.

4.3: Exercise 3

Section **3.3** demonstrated the implementation of the analytical solution to the problem of finding the launch angle of a soccer ball to hit a target at a given distance.

It also contained the code to solve and plot the angles or return error messages.

Below are the results for a ball kicked from the following disances from Mia:

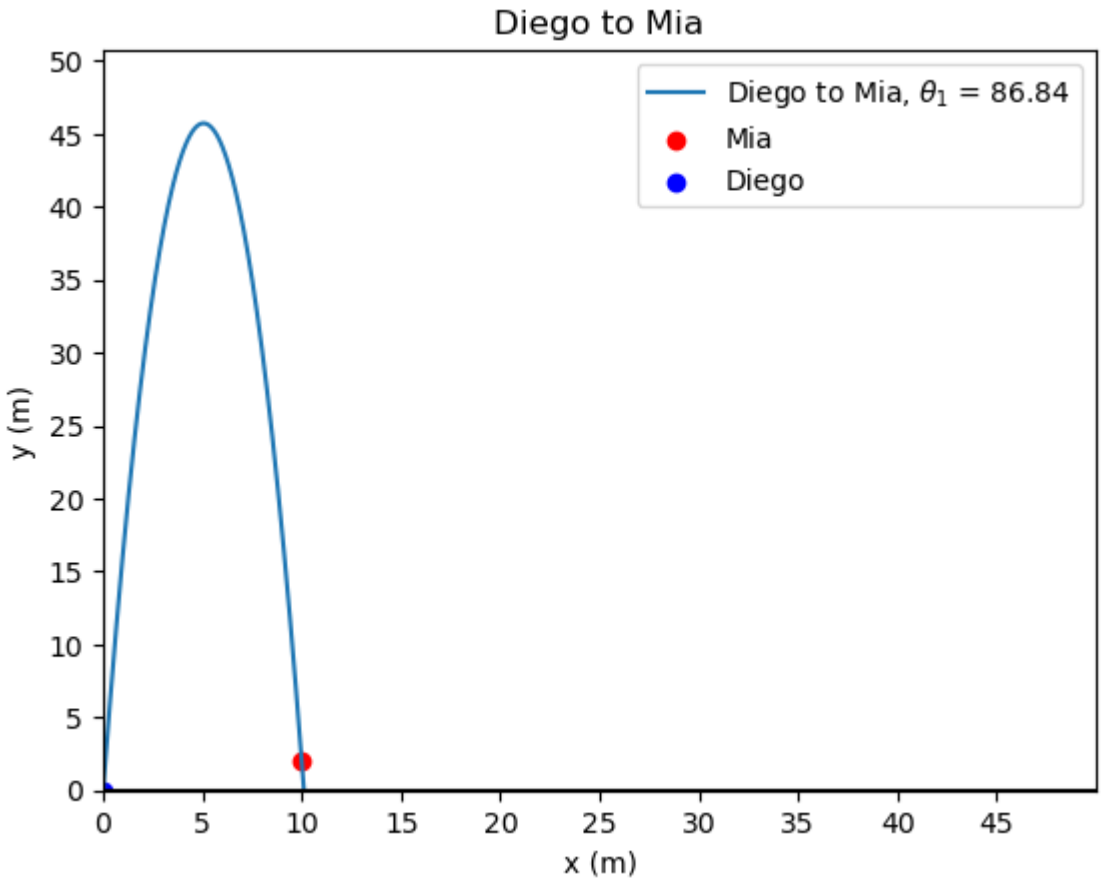


Figure 4.3.1: These conditions only have one solution

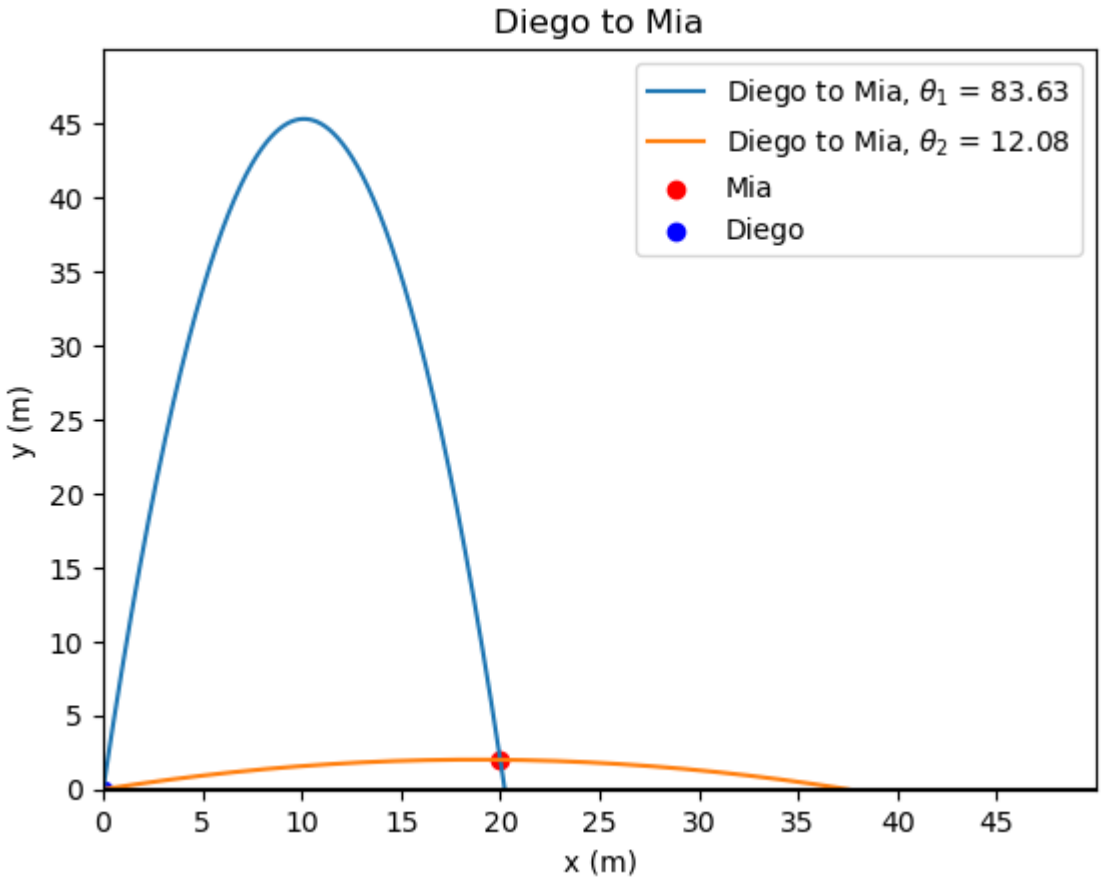


Figure 4.3.2: These conditions have two solutions

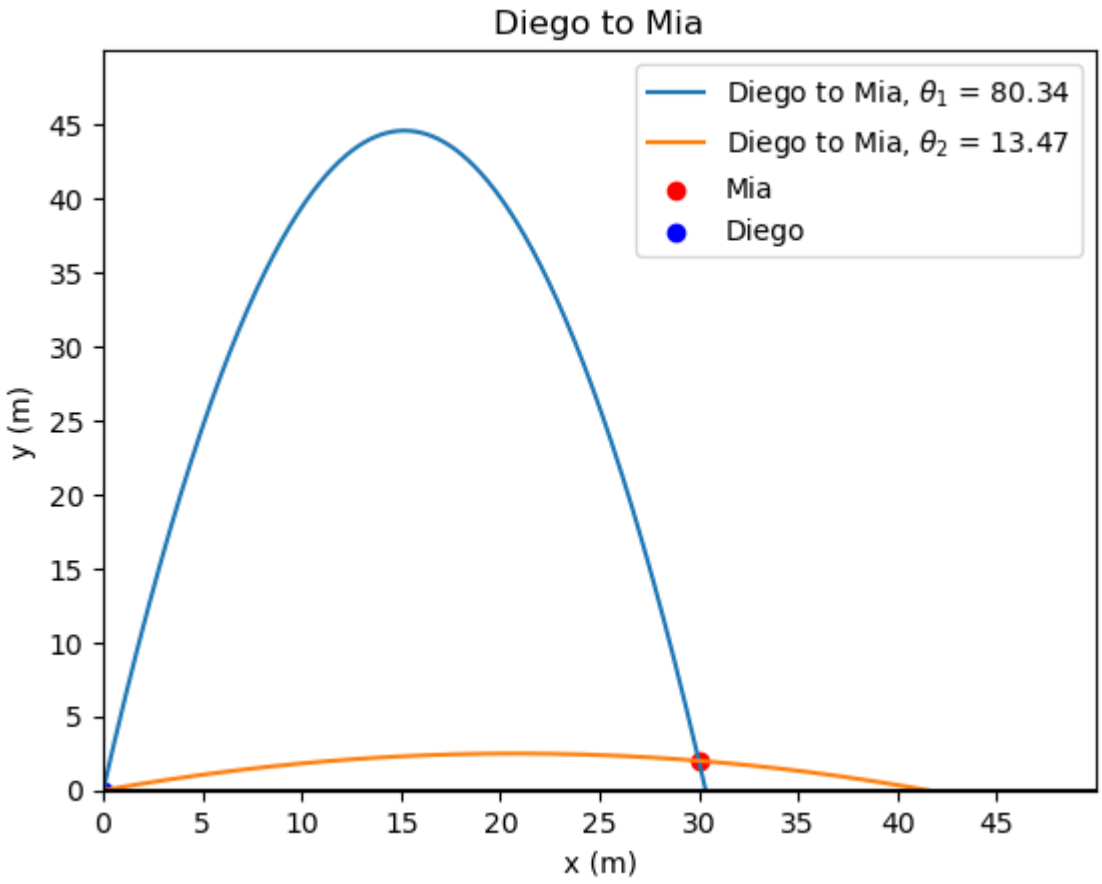


Figure 4.3.3: These conditions have two solutions

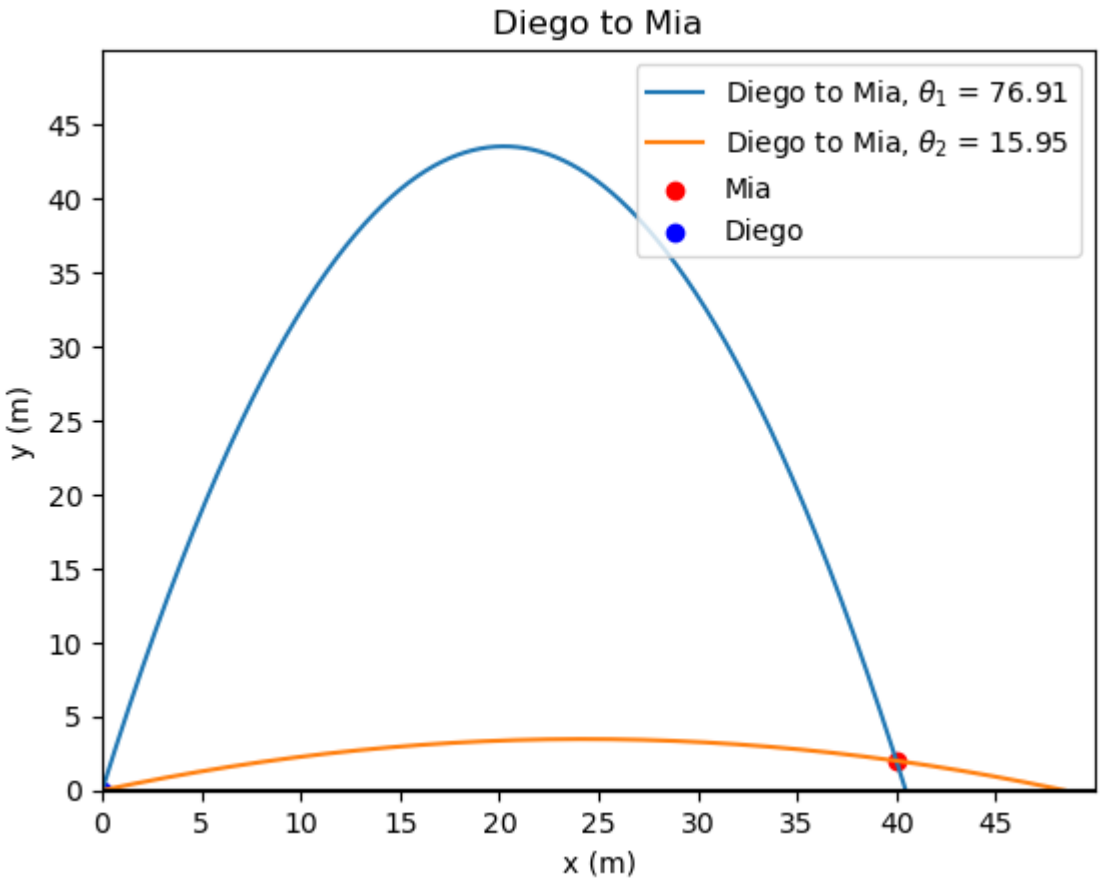
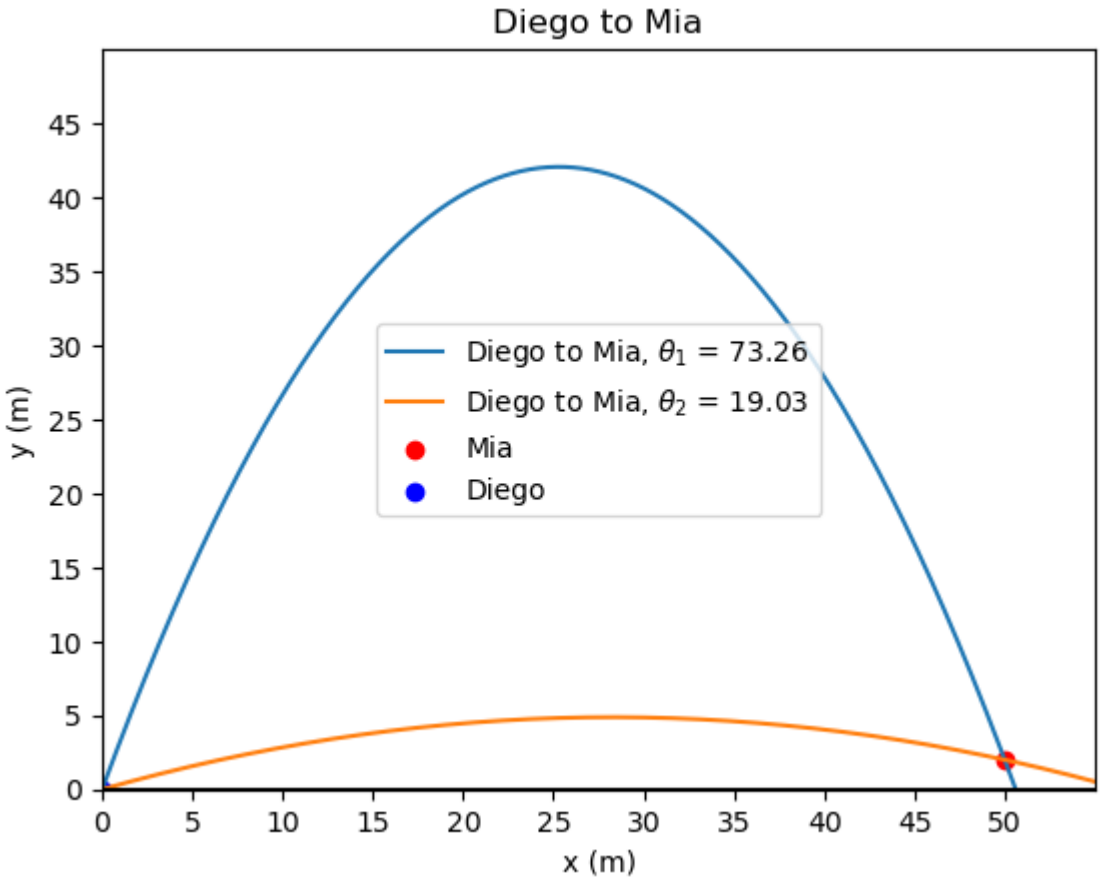
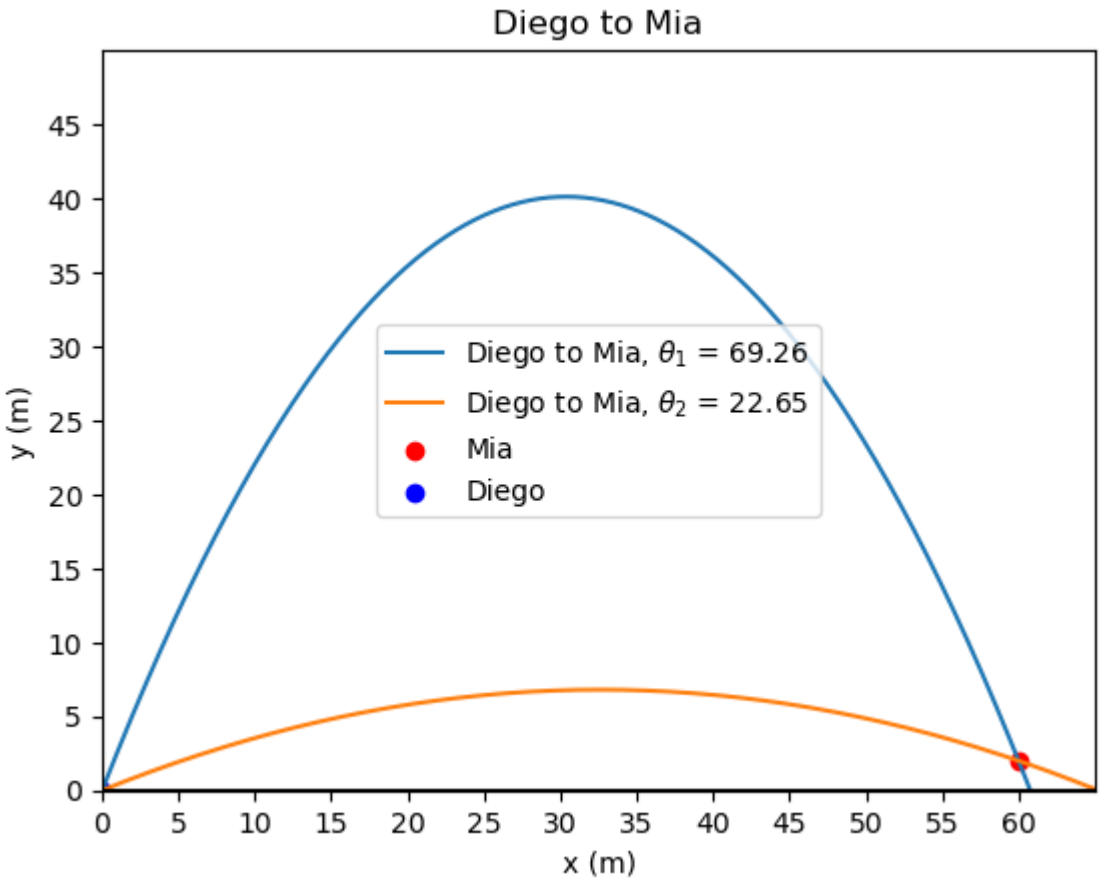


Figure 4.3.4: These conditions have two solutions



> Figure 4.3.5: These conditions have two solutions



> Figure 4.3.6: These conditions have two solutions

```
'no solutions, try a higher velocity. There are no solutions for distance = 100'
```

> Figure 4.3.7: These conditions have no solutions

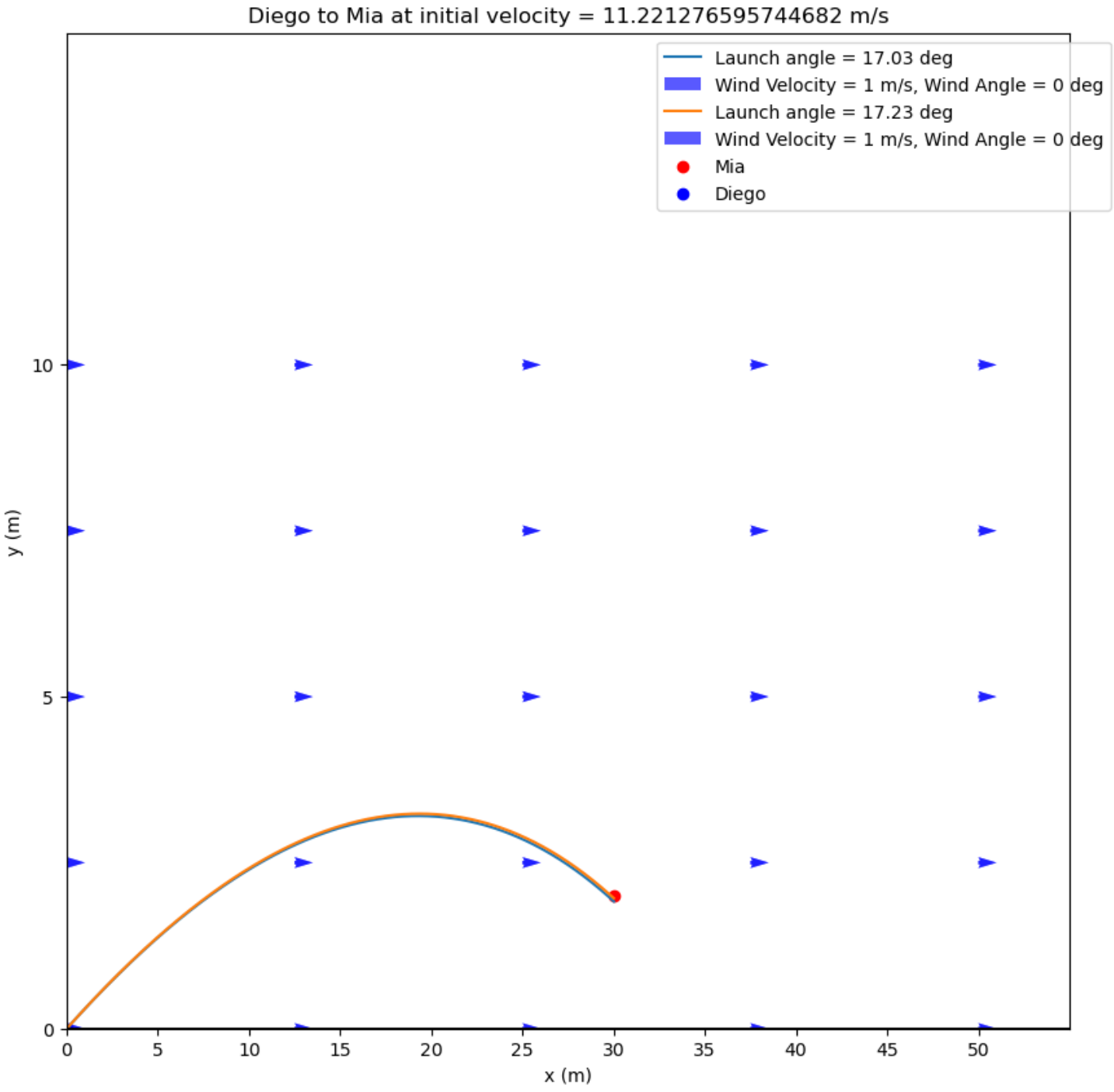
The results above demonstrate the implementation of the analytical solution to the problem of finding the launch angle of a soccer ball to hit a target at a given distance. The results also demonstrate the limitations of the analytical solution.

It is important to note that in real life these distances, and launch angles are not always realistic/plausible for a soccer match

EDIT:

As previously stated in 2.3 and 3.3, I realized that I was not accounting for the air resistance when that is the purpose of the project. Given the time constraints I am working under, I have implemented a coded solution that I believe gives the correct angle given the conditions, however it is found through brute force and a little estimation, and not actual analytical methods. The results are found below. I want to note that Initial for the conditions shown in 4.3.8, there was an similarly a

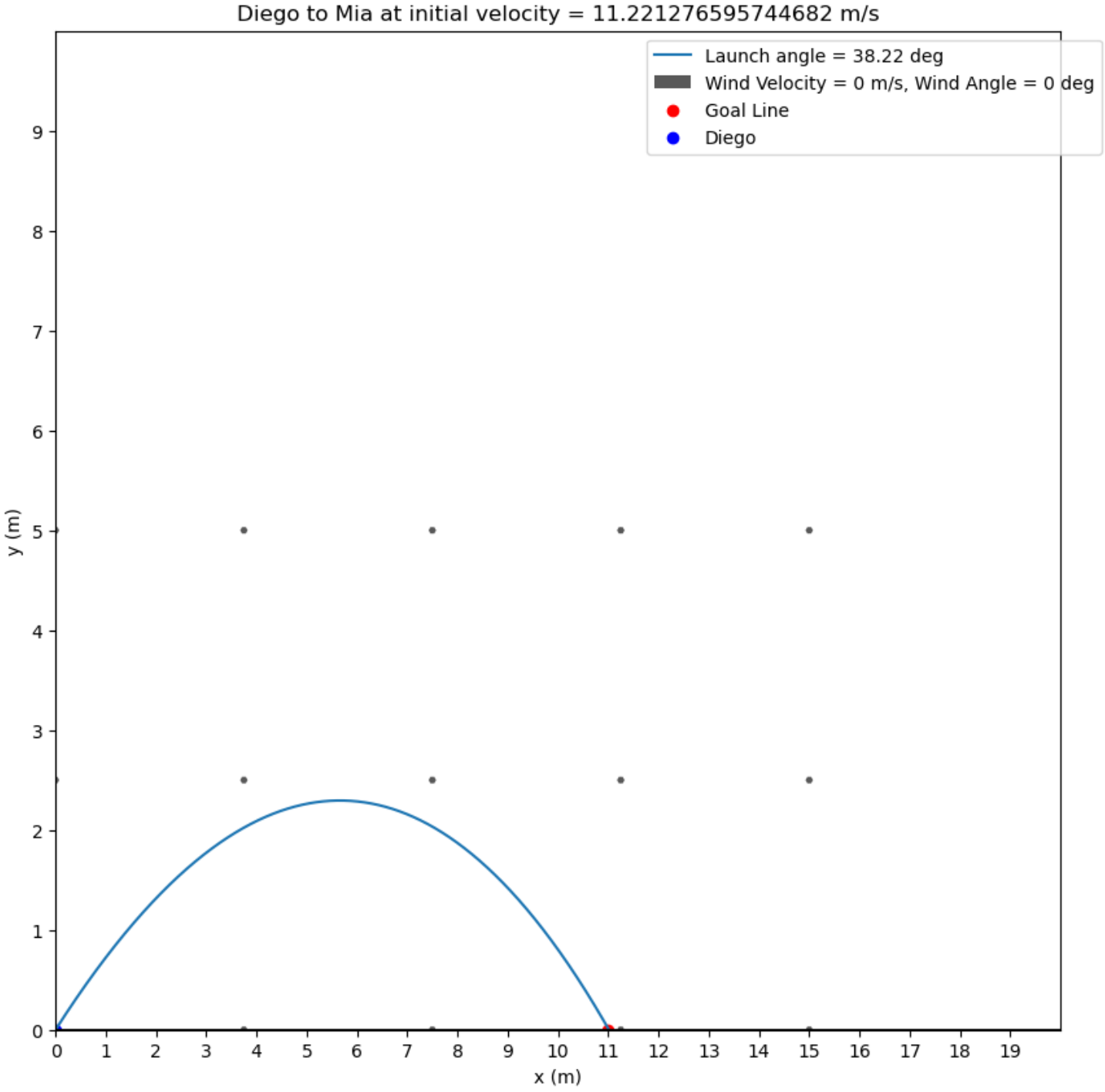
valid angle of around 17°, as well as a valid angle around 71°. Unfortunately I accidentally overrode the file so it is not shown below. However, the code is in the section 3.3.4. For the initial code that generated/would have it.



> Figure 4.3.8: This figure is an example of two angles that each have a valid trajectory to hit Mia's head on it's descent. Both angles are roughly 17°, and reach the target

4.3.4: Exercise 4

As mentioned in **2.4** these values are only approximated and not found analytically. nevertheless, I believe that they are very reasonable values for a spoon shot. The velocity is around $11.2\frac{m}{s}$, well under the average professional kick velocity which makes sense. Similarly the trajectory is resonable, staying under 3 meters for the entire flight. It has a lob, but the angle of around 38.22° is not ridiculous and is certainly realistic for a spoon shot by a professional player. Below is the graph:



5. Conclusion

There are many factors to consider when plotting the trajectory of a soccerball in motion. The flight of a ball is sensitive to many conditions, including the environment it is in, as well as the initial conditions of its launch. Thus, to reach different desired targets in the same environment, these initial conditions must change to compensate for the change in the environment. While these results are good estimations for a soccer ball's trajectory, it is not exhaustive, and in reality there are many more conditions that this report does not consider. For example, we do not talk about the effect of spin, nor the fact that this takes place in a 3D environment, not a 2D environment like these simulations are in. However, these are still a good start for predicting a soccer ball's motion.