Computational Analysis of Air Resistance Effects on Projectile Motion

Luke Keely

Trinity College Dublin,
keelyl@tcd.ie

Abstract. In this lab exercise, we look at the effects of air resistance on projectile motion. Traditional projectile motion in a vacuum is easily predictable, following a parabolic trajectory influenced only by gravity. However, under real-world conditions, air resistance changes this simple path. The magnitude of air resistance relies on the projectile's velocity and physical characteristics, such as shape and density.

This project uses numerical methods, such as Euler's method and the Trapezoidal rule, to solve a system of differential equations that model the motion of spherical projectiles affected by linear and quadratic drag forces.

The study looks at approximation methods to decide if the drag can be simplified based on object speed and size and the effect of air resistance on free-fall time, terminal velocity, and optimal launch angles in two-dimensional motion.

The results show significant differences between the models with and without air resistance, highlighting the need to account for air resistance when applying these solutions. It is essential to use accurate air resistance models under conditions where it has a more significant impact; these conditions include smaller and lighter objects and high velocities.

Table of Contents

1 Introduction

Projectile motion is a core area in classical mechanics. It is simple in a vacuum where the object propelled through space follows a trajectory in the shape of a parabola influenced only by gravity. However, this ideal scenario is rarely found outside introductory physics problems. In the real world, we encounter air resistance, a drag force proportional to and acting in the opposite direction of the body's velocity. The magnitude of this force depends on the object's shape and the fluid it travels through and often significantly affects the shape of the trajectory.
The importance of air resistance in these calculations is not just in the classroom but a major area of study in applications from sports to aerospace engineering.
This experiment looks at cases where air resistance cannot be neglected. Using numerical methods to solve the equations of motion for the models of the projectiles, we can compare the results to find out what conditions are affected the most and look at the path difference when air resistance is neglected[2].

Traditionally, without air resistance, the equation of motion for a projectile is given by

$$\boldsymbol{a}(t) = g\hat{j}.$$

The drag is modeled as

$$\boldsymbol{F} = -f(v) \cdot \boldsymbol{u},$$

where $f(v) = bV + cV^2$, and $\boldsymbol{u}$ is the unit vector in the velocity direction. Now, the equation of motion with air resistance is given by

$$\boldsymbol{a}(t) = g\hat{\boldsymbol{j}} - \frac{1}{m}\boldsymbol{F}.$$

In our study of projectile motion, we use a system of differential equations to model the effects of air resistance accurately

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ -\mu\, v_x \sqrt{v_x^2 + v_y^2} \\ -g - \mu\, v_y \sqrt{v_x^2 + v_y^2} \end{pmatrix}.$$

For simplicity, in this experiment, we assume that all objects are uniform spheres, giving us the following values for the coefficients: $B = 1.6e-4 Ns/m^2$ and $C = 0.25 Ns^2/m^4$.
This system of equations can easily be solved using numerical methods. In this lab, we use the Trapezoidal rule because we found that this method offers a good blend of reliability and computational speed in a previous experiment[4].

In this in-depth analysis of projectile motion and drag forces, we look at when it is acceptable to approximate our model for air resistance, the effects of drag on free-fall time, and how the body's mass comes into play. Then, we look at projections in the 2D plane and how drag affects the optimal launch angle.

2 Methodology

2.1 Selection of Approximation

When working with our air resistance model, we apply linear and quadratic drag terms. We usually do not need to compute both these terms to get a reasonable estimate of the drag. However, the decision on what term to neglect, if any, is based on the speed and size of the object.

The script `term_approx.py` helps us analyze the term that can approximate the drag based on the speed and diameter of our spherical projectile. It uses a threshold set by the user, in this case, 5%, to decide if both the linear drag term bV and the quadratic drag term cV^2 are significant or whether one can be neglected.

```
1
2  if min(bV[idx], cV2[idx]) > max(THRESHOLD_PERCENT/100 * bV[idx],
       THRESHOLD_PERCENT/100 * cV2[idx]):
3              terms_to_use = "Both"
4          else:
5              terms_to_use = "Quadratic" if bV[idx] < cV2[idx] else "
                   Linear"
```

The code calculates the linear and quadratic drag terms and determines if either force is less than the threshold at $D \times V$.
The code is run over several cases with varying conditions:

```
1
2      cases = {
3          'Baseball': {'D': 0.07, 'V_max': 10},
4          'Oil Drop': {'D': 1.5e-6, 'V_max': 1e-4},
5          'Raindrop': {'D': 0.001, 'V_max': 3},
6      }
```

Object	Size	Speed
Baseball	Moderate	High
Oil Drop	Very Small	Very Low
Raindrop	Small	Moderate

Table 1: Comparison of Objects

The script plots for each case that scale the linear and quadratic drag terms with $D \times V$. Each plot contains the linear term, quadratic term, the combined term, and 5% of each around the $D \times V$ of the projectile.

These plots clearly represent what term to use and why we can neglect the other.

2.2 Free Fall Time

Effect of Air Resistance on Free Fall Time

In the script `free_fall.py`, we look at how bodies of increasing densities affect the time it takes to fall from a fixed height under air resistance. The script calculates how long each particle takes to fall with and without air resistance. In this case, we use spheres that are 1cm in diameter. At this size, it is acceptable to neglect the linear term. However, it is left as is due to the short time needed to run. As this system is not very complex and runs over a short period, we use Euler's method to solve for the time taken to reach the ground[3].

```
def dust_particle_motion_with_air_resistance(Vy):
    return -g + (B * D * Vy + C * D**2 * Vy**2)/mass

for density in densities:
    mass = calculate_mass(density)
    time_with_air = simulate_fall_with_air_resistance(mass)
    times_with_air_resistance.append(time_with_air)
```

Terminal Velocity and Time to Terminal Velocity

Terminal velocity is the constant speed achieved by a falling body when the force of air resistance equals the force of gravity. By observing the equation, we can see the mass of the particle will affect the equilibrium between gravity and the drag.

The script `term_velocity.py` uses the trapezoidal rule to iteratively calculate the velocity until the change in velocity becomes negligible. This is when the particle has reached terminal velocity.

```
for step in range(nsteps):
    time = step * dt
    for name, properties in cases.items():
        Vy = properties['Vys'][step-1] if step > 0 else 0
        k1_Vy = dt * (-g + calculate_air_resistance(properties['density
            '], Vy))
        k2_Vy = dt * (-g + calculate_air_resistance(properties['density
            '], (Vy+ k1_Vy)))
        Vy_new = Vy + (k1_Vy + k2_Vy) / 2
        properties['Vys'][step] = Vy_new
        if properties['terminal_velocity'] is None:
            if abs(Vy_new - Vy) <= tolerance:
                properties['terminal_velocity'] = Vy_new
                properties['time_to_terminal'] = time
```

2.3 Trajectories

We move from looking at motion in one dimension to two dimensions. When we calculate the trajectory of a particle neglecting air resistance, we take velocity in the horizontal direction as constant. By introducing drag, we can not do this anymore. The forces are split up in Vx and Vy and calculated separately. We consider the case with and without air resistance and compare their paths for various projection angles.

```
def simulate_projectile_motion(air_resistance_type, theta):
    Vx = V * np.cos(theta)
    Vy = V * np.sin(theta)
    X, Y = 0, 0
    trajectory = []
    while Y >= 0:
        ax, ay = projectile_motion(Vx, Vy, air_resistance_type, theta)
        Vx += ax * dt
        Vy += ay * dt
        X += Vx * dt
        Y += Vy * dt
        trajectory.append((X, Y))
    return trajectory
```

The trajectories are plotted for each launch angle and air resistance condition, showing how air resistance alters the projectile's path.

2.4 Optimum Angles

When air resistance is introduced, the optimum launch angle is no longer 45 degrees in general. The script `optimal_angles.py` looks at how changes in density, diameter, and initial velocity of a projectile influence its optimal launch angle to reach the maximum range.

The maximum distance is then found, and a heatmap is used to visualize the results.

```
def find_optimal_angles(V):
    optimal_angles_matrix = np.zeros((len(densities), len(diameters)))
    for i, mass_row in enumerate(tqdm(masses, desc=f"Velocity {V} m/s")
        ):
        for j, mass in enumerate(mass_row):
            D = diameters[j]
            distances = [simulate_projectile_motion(theta, mass, D, V)
                for theta in angles]
            optimal_angle = angles[np.argmax(distances)]
            optimal_angles_matrix[i, j] = np.degrees(optimal_angle)
    return optimal_angles_matrix
```

The function `simulate_projectile_motion` uses the equation of motion to simulate the trajectories over the range of launch angles. Euler's method was chosen for this due to the large computation volume, but the accuracy of this method gave some issues at higher velocities.

2.5 Software and Libraries

The project was made using Python 3.11.5, running in the Spyder IDE within the Anaconda environment. Several Python libraries were used in making this project. NumPy was essential for numerical operations, while Matplotlib and Seaborn were used to plot the data. The Seaborn library, in particular, was great for data visualization, making it more aesthetic and functional compared to running Matplotlib alone.

2.6 Challenges and Modifications

The main challenge of this experiment was calculating the optimum angle. Due to the large number of computations involved, each plot took a long time to run. This was partly solved by implementing Euler's method to reduce computation time; however, at higher velocities, smaller step sizes were needed, again increasing computation time. At higher velocities, the resolution of the heatmap was reduced from 2500 to 100.

The plotting part of the project had some challenges, mainly when representing the data accurately and visually appealingly. This was solved primarily using the Seaborn library, which offers cleaner and more customizable plots than Matplotlib.

3 Results

3.1 Selection of Approximation

The `term_approx.py` script clearly shows us why each approximation is made.

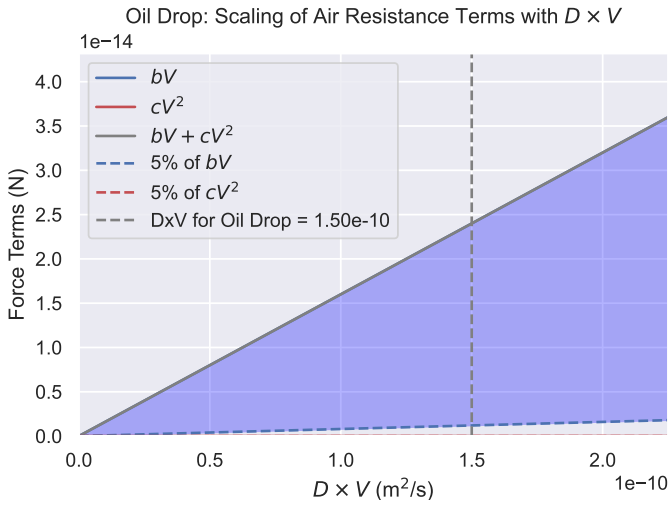


Fig. 3.1: Plot of Oil Drop (D:1.5e-6m, V:1e-4m/s) air resistance terms.

Linear The plot of the drag on the oil drop shows us that when $D \times V$ (D:1.5e-6m, V:1e-4m/s) is very small, the linear term takes over significantly. Here, the quadratic term is negligible compared to the linear term. This suggests that the linear term is much more significant for small particles at low velocities.

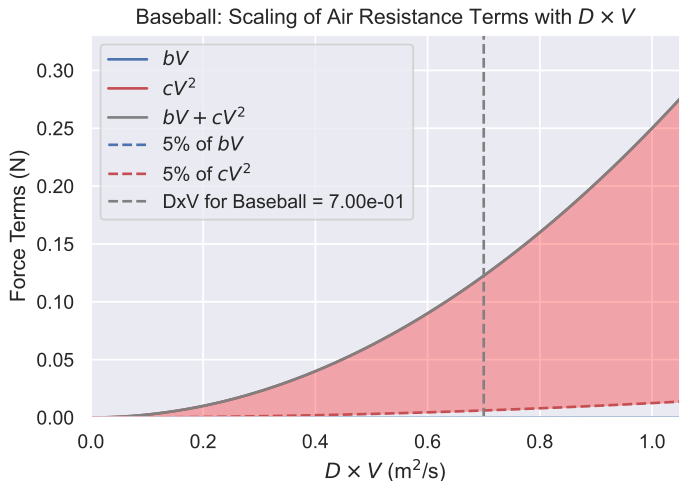


Fig. 3.2: Plot of Baseball (D:0.07m, V:10m/s) air resistance terms.

Quadratic The plot of the drag on the baseball shows us that when $D \times V$ is large, the quadratic term now takes over. Here, the linear term is negligible compared to the quadratic term. This suggests that the quadratic term is much more significant for larger particles at higher velocities.

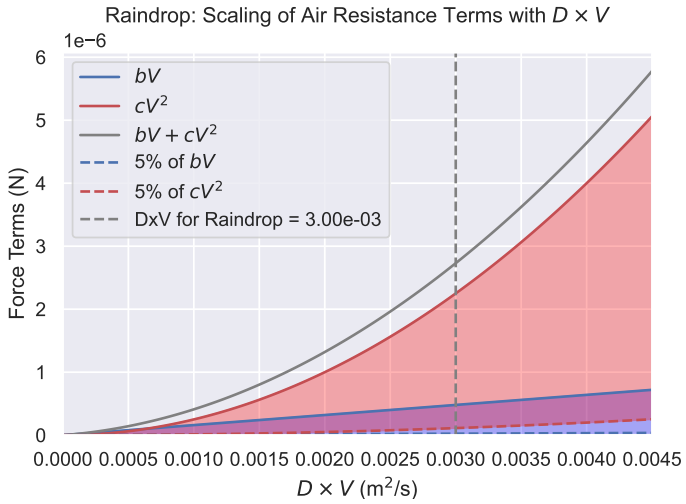


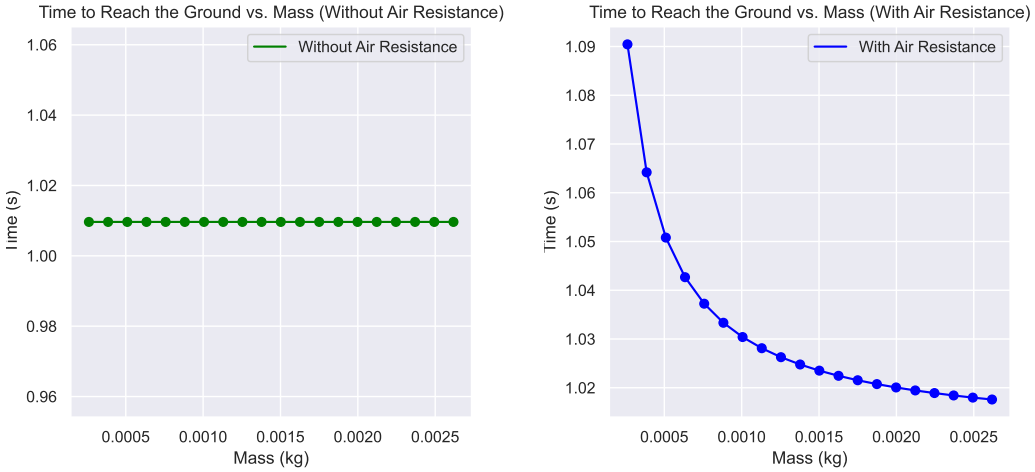
Fig. 3.3: Plot of Raindrop (D:1e-3m, V:3m/s) air resistance terms.

Both The plot of the drag on the raindrop has a crossover point where the quadratic term overtakes the linear term as $D \times V$ increases. The linear term is initially more significant, but the quadratic term overtakes as $D \times V$ grows. Both terms are above the 5% threshold, indicating that neither term can be neglected for accurate calculations at the given $D \times V$.

3.2 Free Fall Time

Effect of Air Resistance on Free Fall Time

The plot without air resistance shows a constant time to reach the ground regardless of the object's mass. This is expected as the only force acting on the body without drag is gravity, which acts on all objects at the same rate.
However, the plot with air resistance shows that the time taken to reach the ground decreases as the mass increases. The plot is non-linear and suggests that as mass tends to infinity, the time to reach the ground will tend to be at a constant value without air resistance.



(a) No Drag

(b) Drag

Fig. 3.4: Plots of Mass vs Freefall time

These graphs show that air resistance significantly affects the fall time of lighter objects, and air resistance may only be neglected in very dense objects.

Mass (kg)	Time with Air Resistance (s)	Time without Air Resistance (s)
0.00038581	1.06419	1.00964
0.00063383	1.04268	1.00964
0.00088185	1.03333	1.00964
0.00112987	1.02811	1.00964
0.00137789	1.02477	1.00964
0.00162591	1.02245	1.00964
0.00187393	1.02075	1.00964
0.00212195	1.01945	1.00964
0.00236997	1.01842	1.00964
0.00261799	1.01759	1.00964

Table 2: Experimental Data (Every Second Row, Rounded to 5 Decimal Places)

Terminal Velocity and Time to Terminal Velocity

This plot shows how particles of different masses achieve their terminal velocities over time. Each curve represents a different particle, from very light to heavy, with the horizontal lines indicating their terminal velocity.

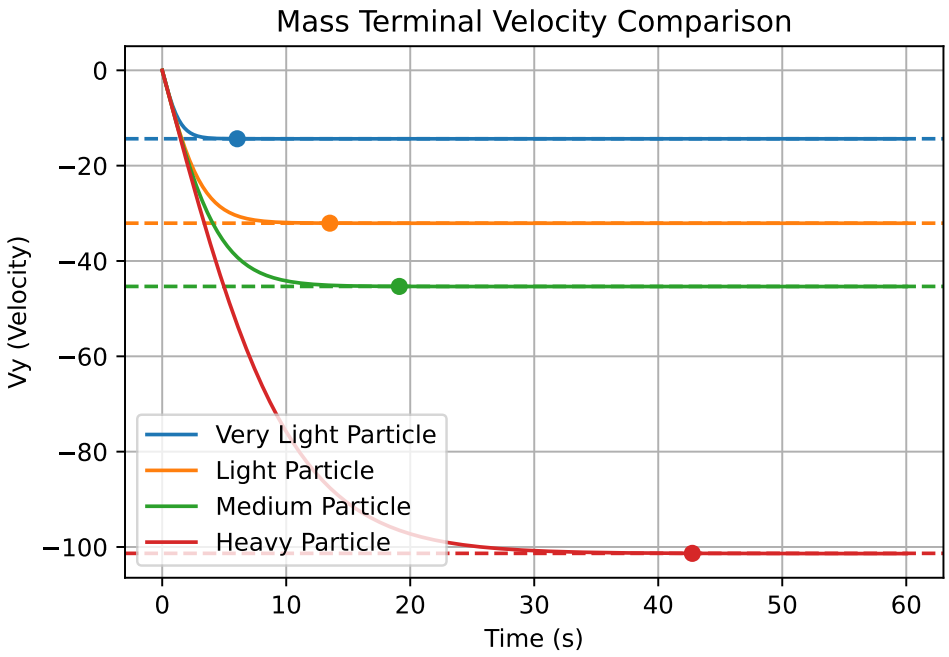


Fig. 3.5: Terminal Velocity and Time to Terminal Velocity for Various Particle Densities

For the very light particle, shown by the blue line, we can see it takes a short time to reach a relatively low terminal velocity, suggesting that air resistance significantly affects lighter particles. As we move to the heavy particle, shown by the red line, we can see that it takes the longest to reach its terminal velocity and has the highest among the four. This further suggests that air resistance has a much lesser effect on denser bodies.

Particle	**Density** (kg/m^3)	**Diameter** (m)	**Terminal V** (m/s)	**Time** (s)
Very Light Particle	1000	0.01	-14.36	6.048
Light Particle	5000	0.01	-32.07	13.52
Medium Particle	10000	0.01	-45.34	19.119
Heavy Particle	50000	0.01	-101.34	42.748

Table 3: Physical Characteristics of Particles

3.3 Trajectories

The plots show us how air resistance affects the path of a projectile at different launch angles: 30 degrees, 45 degrees, and 60 degrees. Each plot has four marked trajectories: no air resistance, linear air resistance, quadratic air resistance, and combined air resistance.

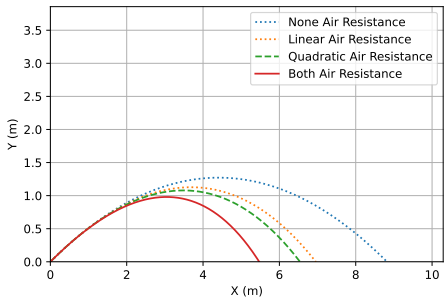


Fig. 3.6: Plot of Trajectory: 10 m/s, 30°

At this angle, the trajectory with no air resistance reaches farther than the ones with air resistance. The linear air resistance causes a slight reduction in range, while the quadratic resistance has a more significant impact, shortening the range further. The line representing the full term shows the shortest range, indicating that air resistance has a significant effect even at lower launch angles.

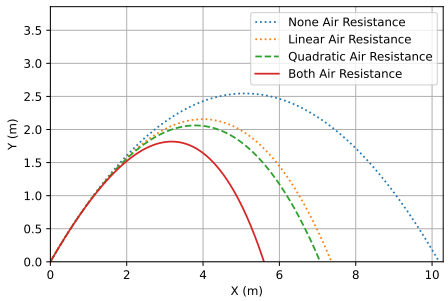


Fig. 3.7: Plot of Trajectory: 10 m/s, 45°

This is the ideal launch angle without air resistance and has the longest range. However, this is not the case when air resistance is considered. Both the height and range decrease significantly. The distance is close to the previous plot, with the drag suggesting that the optimum angle lies between 30° and 45°.

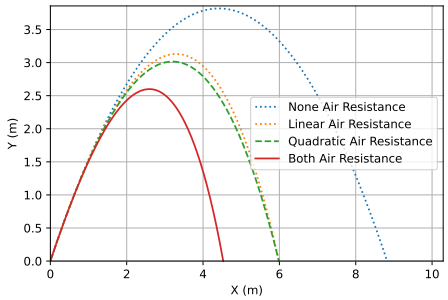


Fig. 3.8: Plot of Trajectory: 10 m/s, 60°

At the highest launch angle, the trajectories reach the highest heights. However, air resistance causes the projectile to peak lower and land sooner. This added height also causes the projectile to fall short in the case with air resistance but matches the 30° case without drag taken into account.

3.4 Optimum Angles

The heatmap plots show the optimal launch angle as a function of the density and diameter of the projectile over different launch velocities (1 m/s, 5 m/s, and 10 m/s).
Each heatmap shows the color change, displaying how the optimal launch angle changes with the projectile's density and diameter.

We can generally see ring-like shapes centered at the bottom right of constant optimum angles, with the largest and heaviest projectiles being least influenced by drag forces and closest to 45 degrees. This suggests that both properties can individually influence the optimum angle.

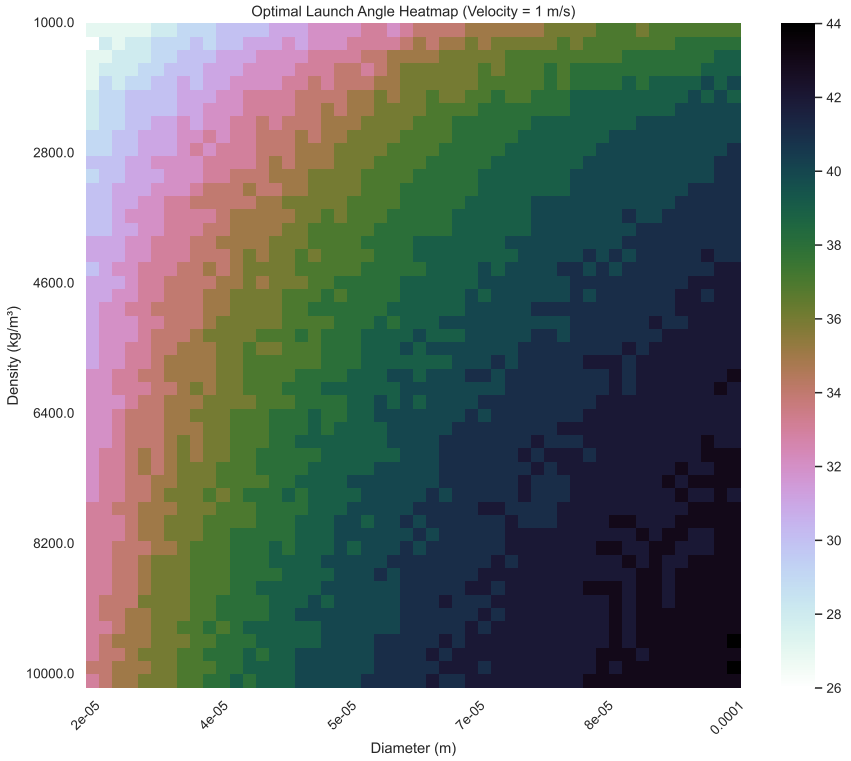


Fig. 3.9: Optimum Angle in terms of diameter and density: 1m/s

1m/s At low initial velocities, the optimal angles tend to be higher, suggesting that at lower velocities, the projectile's optimal angle would be closer to the classic 45 degrees.

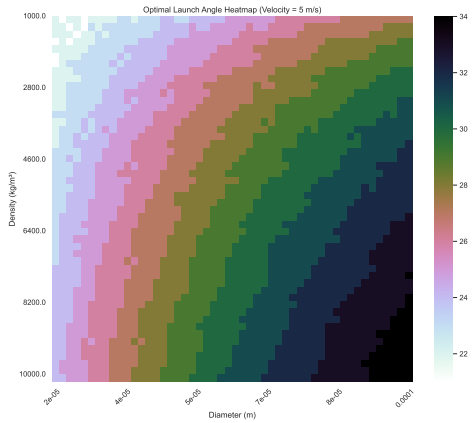


Fig. 3.10: Optimum Angle in terms of diameter and density: 5m/s

5m/s As the velocity increases, the optimal launch angles decrease. This is especially noticeable for lighter and smaller projectiles. This implies a more significant impact of air resistance at this velocity.

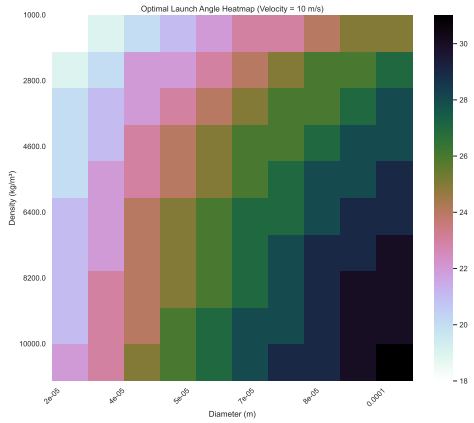


Fig. 3.11: Optimum Angle in terms of diameter and density: 10m/s

10m/s At higher velocities, the optimal launch angles decrease even further, again showing the increased influence of drag. The highest velocity of the group has the smallest range or angles.

4 Conclusion

This computational experiment provided a detailed investigation into the effects of air resistance on projectile motion, helping us to understand the relationships between drag forces and the physical characteristics of the projectile. The study confirms that is crucial to take into account the effect that air resistance has on trajectories, free-fall times, and terminal velocities. Depending on the object, without taking these forces into account, calculations are likely to be highly inaccurate.

Using several computational models, we have shown that linear and quadratic air resistance terms are important to accurately predict projectile motion. The assumption of neglecting air resistance leads to significant differences in the calculated paths, and our models of air resistance must be accurate when simulating problems for real-world applications.

For bodies in free fall, we saw a clear relationship between the body's mass and the time to reach the ground, as opposed to the mass-independent behavior in a vacuum. This mass-dependent behavior is further shown in that air resistance affects larger objects less through the time it takes to reach terminal velocity, where the time taken increases with the body's density.

Our trajectory analysis showed how air resistance affects the path of a projectile, with the optimal launch angle moving away from 45 degrees. This change becomes more significant as the projectile's speed increases.

Lastly, the heatmaps provide a visual tool to help us understand how different factors affect the optimal launch angle. The heavier and larger objects get closer to the case without air resistance, especially at lower speeds.

In conclusion, this experiment provides a well-rounded overview of air resistance acting on projectiles in computational physics. Future research might include more complex shapes and uneven surfaces to be closer to real-world applications.

References

1. Department of Physics, Trinity College Dublin, *SF Computational Lab Handbook.*
2. Wikipedia, *Projectile motion.*
3. Freecodecamp.org, *Euler's Method Explained with Examples.*
4. Wikipedia, *Trapezoidal rule (differential equations).*
5. Documentation for NumPy, Matplotlib, and Seaborn.
6. Python Software Foundation, *Python Documentation.*

A Code

A.1 term_approx.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

# Constants
B = 1.6e-4  # N s/m^2
C = 0.25  # N s^2/m^4
THRESHOLD_PERCENT = 5  # percentage below which a term is considered negligible

def create_subfolder():
    subfolder_name = "term_approx"
    if not os.path.exists(subfolder_name):
        os.makedirs(subfolder_name)
    return subfolder_name

def plot_case(D, V_max, case_name, DxV_case, subfolder):
    sns.set_theme()
    V = np.linspace(0, V_max * 1.5, 100)
    bV = B * D * V
    cV2 = C * D**2 * V**2
    total = bV + cV2
    DxV = D * V

    plt.figure(figsize=(6, 4), dpi=300)
    plt.plot(DxV, bV, 'b-', label='$bV$')
    plt.plot(DxV, cV2, 'r-', label='$cV^2$')
    plt.plot(DxV, total, color='gray', linestyle='-', label='$bV + cV^2$')
    plt.plot(DxV, THRESHOLD_PERCENT/100 * bV, 'b--', label=f'{THRESHOLD_PERCENT}% of $bV$')
    plt.plot(DxV, THRESHOLD_PERCENT/100 * cV2, 'r--', label=f'{THRESHOLD_PERCENT}% of $cV^2$')
    plt.fill_between(DxV, bV, THRESHOLD_PERCENT/100 * bV, color='blue', alpha=0.3)
    plt.fill_between(DxV, cV2, THRESHOLD_PERCENT/100 * cV2, color='red', alpha=0.3)

    plt.axvline(DxV_case, color='gray', linestyle='--', label=f'DxV for {case_name} = {DxV_case:.2e}')

    plt.xlim(0, V_max * D * 1.5)
    plt.ylim(0, max(max(bV), max(cV2)) * 1.2)

    plt.xlabel('$D \\times V$ (m$^2$/s)')
    plt.ylabel('Force Terms (N)')
    plt.title(f'{case_name}: Scaling of Air Resistance Terms with $D \\times V$')
    plt.legend()
    plt.grid(True)
    plt.savefig(os.path.join(subfolder, f"{case_name}_force_terms_scaling.png"))
```

```
    plt.savefig(os.path.join(subfolder, f"{case_name}_force_terms_scaling.pdf"))
    plt.show()


subfolder = create_subfolder()
cases = {
    'Baseball': {'D': 0.07, 'V_max': 10},
    'Oil Drop': {'D': 1.5e-6, 'V_max': 1e-4},
    'Raindrop': {'D': 0.001, 'V_max': 3},
}


data_to_save = []

for name, params in cases.items():
    D, V_max = params['D'], params['V_max']
    DxV_case = D * V_max
    plot_case(D, V_max, name, DxV_case, subfolder)

    V = np.linspace(0, V_max, 100)
    bV = B * D * V
    cV2 = C * D**2 * V**2
    idx = np.argmin(np.abs(D * V - DxV_case))

    if min(bV[idx], cV2[idx]) > max(THRESHOLD_PERCENT/100 * bV[idx], THRESHOLD_PERCENT/100 * cV2[idx]):
        terms_to_use = "Both"
    else:
        terms_to_use = "Quadratic" if bV[idx] < cV2[idx] else "Linear"

    data_to_save.append({
        'Case': name,
        'D': D,
        'V_max': V_max,
        'DxV_case': DxV_case,
        'Terms to Use': terms_to_use,
        'bV_at_DxV': bV[idx],
        'cV2_at_DxV': cV2[idx]
    })

df = pd.DataFrame(data_to_save)
print(df)
csv_file = os.path.join(subfolder, "specific_cases.csv")
df.to_csv(csv_file, index=False)
print(f"Data saved to: {csv_file}")
```

A.2 free_fall.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Constants
B = 1.6e-4  # N s/m^2
C = 0.25
g = 9.81  # Acceleration due to gravity (m/s^2)
dt = 1e-5  # Time step (s)
D = 1e-2  # Diameter of the particle
H = 5  # Initial height (m)
densities = np.linspace(500, 5000, 20)  # Range of densities (kg/m^3) for 20 particles

def calculate_mass(density):
    radius = D / 2
    volume = (4/3) * np.pi * radius**3
    return density * volume

def dust_particle_motion_with_air_resistance(Vy):
    return -g + (B * D * Vy + C * D**2 * Vy**2)/mass

def simulate_fall_with_air_resistance():
    Vy = 0
    Y = H
    time = 0

    while Y > 0:
        Vy += dt * dust_particle_motion_with_air_resistance(Vy)
        Y += Vy * dt
        time += dt

    return time

def simulate_fall_without_air_resistance():
    return np.sqrt(2 * H / g)

# Prepare directory for plots
plot_dir = 'falling_particle_plots'
os.makedirs(plot_dir, exist_ok=True)

# Calculate time to reach the ground for different densities
times_with_air_resistance = []
times_without_air_resistance = []
masses = []

for density in densities:
    mass = calculate_mass(density)
    time_with_air = simulate_fall_with_air_resistance()
    time_without_air = simulate_fall_without_air_resistance()
    times_with_air_resistance.append(time_with_air)
    times_without_air_resistance.append(time_without_air)
    masses.append(mass)

```

```
# With Air Resistance
plt.figure(figsize=(5, 5), dpi=300)
plt.plot(masses, times_with_air_resistance, label='With Air Resistance', color='blue')
plt.scatter(masses, times_with_air_resistance, color='blue')
plt.xlabel('Mass (kg)')
plt.ylabel('Time (s)')
plt.title('Time to Reach the Ground vs. Mass (With Air Resistance)')
plt.grid(True)
plt.legend()
plt.savefig(os.path.join(plot_dir, 'Time_to_Ground_With_Air_Resistance.png'))
plt.savefig(os.path.join(plot_dir, 'Time_to_Ground_With_Air_Resistance.pdf'))
plt.show()

# Without Air Resistance
plt.figure(figsize=(5, 5), dpi=300)
plt.plot(masses, times_without_air_resistance, label='Without Air Resistance', color='green')
plt.scatter(masses, times_without_air_resistance, color='green')
plt.xlabel('Mass (kg)')
plt.ylabel('Time (s)')
plt.title('Time to Reach the Ground vs. Mass (Without Air Resistance)')
plt.grid(True)
plt.legend()
plt.savefig(os.path.join(plot_dir, 'Time_to_Ground_Without_Air_Resistance.png'))
plt.savefig(os.path.join(plot_dir, 'Time_to_Ground_Without_Air_Resistance.pdf'))
plt.show()

data = {
    'Mass (kg)': masses,
    'Time with Air Resistance (s)': times_with_air_resistance,
    'Time without Air Resistance (s)': times_without_air_resistance
}
df = pd.DataFrame(data)
print(df)
data_filename = os.path.join(plot_dir, 'particle_fall_times.csv')
df.to_csv(data_filename, index=False)
```

A.3 term_velocity.py

```
import os
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Constants
B = 1.6e-4  # N s/m^2
C = 0.25    # N s/m^2
g = 9.81  # Acceleration due to gravity (m/s^2)
dt = 1e-3  # Time step (s)
D = 1e-2  # Constant diameter for all cases
tolerance = 1e-5
t_max = 60  # Maximum simulation time (s)
nsteps = int(t_max / dt)

# Function to calculate air resistance
def calculate_air_resistance(density, velocity):
    radius = D / 2
    volume = (4/3) * np.pi * radius**3
    mass = density * volume
    b = B * D
    c = C * D**2
    return (b * velocity + c * velocity*velocity)/mass

# cases
cases = {
    'Very Light Particle': {'density': 1e3, 'Vys': np.zeros(nsteps), '
        terminal_velocity': None, 'time_to_terminal': None},
    'Light Particle': {'density': 5e3, 'Vys': np.zeros(nsteps), '
        terminal_velocity': None, 'time_to_terminal': None},
    'Medium Particle': {'density': 1e4, 'Vys': np.zeros(nsteps), '
        terminal_velocity': None, 'time_to_terminal': None},
    'Heavy Particle': {'density': 5e4, 'Vys': np.zeros(nsteps), '
        terminal_velocity': None, 'time_to_terminal': None},
}

for step in range(nsteps):
    time = step * dt
    for name, properties in cases.items():
        Vy = properties['Vys'][step-1] if step > 0 else 0
        k1_Vy = dt * (-g + calculate_air_resistance(properties['density
            '], Vy))
        k2_Vy = dt * (-g + calculate_air_resistance(properties['density
            '], (Vy+ k1_Vy)))
        Vy_new = Vy + (k1_Vy + k2_Vy) / 2
        properties['Vys'][step] = Vy_new
        if properties['terminal_velocity'] is None:
            if abs(Vy_new - Vy) <= tolerance:
                properties['terminal_velocity'] = Vy_new
                properties['time_to_terminal'] = time

plot_dir = 'particle_motion_plots'
os.makedirs(plot_dir, exist_ok=True)

plt.figure(figsize=(6, 4), dpi=300)
```

```
times = np.arange(0, dt * nsteps, dt)
line_colors = {}

for title, properties in cases.items():
    line, = plt.plot(times, properties['Vys'], label=title)
    line_colors[title] = line.get_color()
    if properties['terminal_velocity'] is not None:
        plt.axhline(y=properties['terminal_velocity'], color=
            line_colors[title], linestyle='--')
        plt.plot(properties['time_to_terminal'], properties['
            terminal_velocity'], 'o', color=line_colors[title])
    df = pd.DataFrame({'Time (s)': times, 'Velocity (m/s)': properties[
        'Vys']})
    filename = f"{plot_dir}/{title.replace(' ', '_')}_data.csv"
    df.to_csv(filename, index=False)
    print(f"{title} - Density: {properties['density']} kg/m^3, Terminal
         Velocity: {properties['terminal_velocity']} m/s, Time to
        Terminal: {properties['time_to_terminal']} s")

plt.xlabel('Time (s)')
plt.ylabel('Vy (Velocity)')
plt.title('Mass Terminal Velocity Comparison')
plt.legend()
plt.grid(True)
plt.savefig(f"{plot_dir}/combined_particle_motion_plot.png")
plt.savefig(f"{plot_dir}/combined_particle_motion_plot.pdf")
plt.show()
```

A.4 trajectory.py

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os

# Constants
B = 1.6e-4  # N s/m^2 (linear drag coefficient)
C = 0.25    # N s^2/m^4 (quadratic drag coefficient)
g = 9.81    # Acceleration due to gravity (m/s^2)
dt = 1e-3   # Time step (s)
D = 1e-4    # Diameter of the projectile
density = 8e4  # Density of the projectile (kg/m^3)
V = 10      # Launch velocity (m/s)
angles_degrees = [30, 45, 60]  # Small, medium, and large angles

radius = D / 2
volume = (4/3) * np.pi * radius**3
mass = density * volume


def projectile_motion(Vx, Vy, air_resistance_type, theta):
    if air_resistance_type == 'none':
        ax, ay = 0, -g
    else:
        bV = B * D if air_resistance_type in ['linear', 'both'] else 0
        cV2 = C * D**2 if air_resistance_type in ['quadratic', 'both']
            else 0
        total_velocity = np.sqrt(Vx**2 + Vy**2)
        air_resistance_force = (bV + cV2 * total_velocity) / mass
        ax = -air_resistance_force * Vx
        ay = -g - air_resistance_force * Vy
    return ax, ay

def simulate_projectile_motion(air_resistance_type, theta):
    Vx = V * np.cos(theta)
    Vy = V * np.sin(theta)
    X, Y = 0, 0
    trajectory = []
    while Y >= 0:
        ax, ay = projectile_motion(Vx, Vy, air_resistance_type, theta)
        Vx += ax * dt
        Vy += ay * dt
        X += Vx * dt
        Y += Vy * dt
        trajectory.append((X, Y))
    return trajectory

output_dir = "trajectory"
os.makedirs(output_dir, exist_ok=True)

# Calculate the maximum X and Y values across all trajectories
max_X, max_Y = 0, 0
for angle in angles_degrees:
    theta = np.radians(angle)
    for resistance_type in ['none', 'linear', 'quadratic', 'both']:
```

```
        trajectory = simulate_projectile_motion(resistance_type, theta)
        max_X = max(max_X, max(trajectory, key=lambda x: x[0])[0])
        max_Y = max(max_Y, max(trajectory, key=lambda x: x[1])[1])

max_X += max_X / 100
max_Y += max_Y / 100

# Create separate plots for each launch angle
for angle in angles_degrees:
    fig, ax = plt.subplots(figsize=(6, 4), dpi=300)
    theta = np.radians(angle)
    trajectory_data = []

    for resistance_type in ['none', 'linear', 'quadratic', 'both']:
        trajectory = simulate_projectile_motion(resistance_type, theta)

        for x, y in trajectory:
            trajectory_data.append([angle, resistance_type, x, y])
        ax.plot(*zip(*trajectory), label=f'{resistance_type.capitalize()} Air Resistance',
                linestyle='-' if resistance_type == 'both' else '--' if resistance_type == 'quadratic' else ':')
        ax.set_xlim(0, max_X)
        ax.set_ylim(0, max_Y)

    ax.set_xlabel('X (m)')
    ax.set_ylabel('Y (m)')
    ax.legend()
    ax.grid(True)

    df = pd.DataFrame(trajectory_data, columns=['Angle', 'Air Resistance', 'X', 'Y'])
    df.to_csv(f"{output_dir}/trajectory_data_angle_{angle}.csv", index=False)

    plot_filename = f"{output_dir}/trajectory_plot_angle_{angle}"
    plt.savefig(f"{plot_filename}.png")
    plt.savefig(f"{plot_filename}.pdf")
    plt.show()
```

A.5 optimum_angle.py

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from tqdm import tqdm
import os

# Constants
B = 1.6e-4  # Linear drag coefficient (N s/m^2)
C = 0.25    # Quadratic drag coefficient (N s^2/m^4)
g = 9.81    # Acceleration due to gravity (m/s^2)
dt = 1e-5  # Time step (s)

densities = np.linspace(1000, 10000, 10)  # Density range
diameters = np.linspace(2e-5, 1e-4, 10)  # Diameter range
velocities = [1, 5, 10]  # Test velocities

angles = np.radians(np.linspace(1, 89, 89))  # Launch angles from 1 to 89 degrees
masses = np.array([density * (4/3) * np.pi * (D/2)**3 for density in densities for D in diameters]).reshape(len(densities), len(diameters))

output_dir = "optimum_angle"
os.makedirs(output_dir, exist_ok=True)
sns.set_theme()
collected_data = []

def projectile_motion_with_air_resistance(Vx, Vy, D, mass):
    total_velocity = np.sqrt(Vx**2 + Vy**2)
    bV = B * D * total_velocity
    cV2 = C * D**2 * total_velocity**2
    total_drag = bV + cV2
    air_resistance_force = total_drag / mass
    ax = -air_resistance_force * Vx
    ay = -g - air_resistance_force * Vy
    return ax, ay

def simulate_projectile_motion(theta, mass, D, V):
    Vx = V * np.cos(theta)
    Vy = V * np.sin(theta)
    X, Y = 0, 0
    while Y >= 0:
        ax, ay = projectile_motion_with_air_resistance(Vx, Vy, D, mass)
        Vx += ax * dt
        Vy += ay * dt
        X += Vx * dt
        Y += Vy * dt
    return X

def find_optimal_angles(V):
    optimal_angles_matrix = np.zeros((len(densities), len(diameters)))
    for i, mass_row in enumerate(tqdm(masses, desc=f"Velocity {V} m/s")):
        for j, mass in enumerate(mass_row):
```

```
            D = diameters[j]
            distances = [simulate_projectile_motion(theta, mass, D, V)
                for theta in angles]
            optimal_angle = angles[np.argmax(distances)]
            optimal_angles_matrix[i, j] = np.degrees(optimal_angle)
    return optimal_angles_matrix

for V in velocities:
    optimal_angles_matrix = find_optimal_angles(V)

    for i, density in enumerate(densities):
        for j, diameter in enumerate(diameters):
            collected_data.append([V, density, diameter,
                optimal_angles_matrix[i, j]])

    plt.figure(figsize=(12, 10))
    ax = sns.heatmap(optimal_angles_matrix, cmap="cubehelix_r", vmin=np
        .min(optimal_angles_matrix), vmax=np.max(optimal_angles_matrix))
    ax.set_title(f'Optimal Launch Angle Heatmap (Velocity = {V} m/s)')
    ax.set_xlabel('Diameter (m)')
    ax.set_ylabel('Density (kg/m )')
    plt.xticks(ticks=np.linspace(0, len(diameters)-1, 6), labels=np.
        round(np.linspace(diameters.min(), diameters.max(), 6), 5))
    plt.yticks(ticks=np.linspace(0, len(densities)-1, 6), labels=np.
        round(np.linspace(densities.min(), densities.max(), 6), 0))
    plt.xticks(rotation=45)
    plt.yticks(rotation=0)
    plt.savefig(f'{output_dir}/heatmap_velocity_{V}.png')
    plt.savefig(f'{output_dir}/heatmap_velocity_{V}.pdf')
    plt.show()

columns = ['Velocity', 'Density', 'Diameter', 'Optimal_Angle']
collected_data_df = pd.DataFrame(collected_data, columns=columns)
collected_data_df.to_csv(f"{output_dir}/optimum_angle_data.csv", index=
    False)
```