PHYS 580 - Computational Physics

Computational Physics by *Nicholas J. Giordano, Hisao Nakanishi*
Student: **Ralph Razzouk**

Lab 2

Problem 1
For a cannon shell projectile, take a fixed angle of projection, say, 45°, and the initial velocity, say, 700 m/s, and graphically show the differences in trajectories for the following different models:

(a) gravity only, no air drag,

(b) air drag with constant air density,

(c) air drag with the isothermal model of air density, and

(d) air drag with the adiabatic model, using $\gamma = 1.4$.

Set $\mathrm{B_2/m} = 4 \times 10^{-5}$ (in 1/m units) at ground level. The adiabatic model is where

$$|\mathrm{F_{drag}}| = \mathrm{B_2}\left(1 - \frac{ay}{\mathrm{T_{grd}}}\right)^{\frac{1}{\gamma-1}} v^2$$

(a is the rate of temperature decrease per rise from sea level. Use $a = 6.5\times10^{-3}$ K/m and $T_{\mathrm{grd}} = 293$ K). Can you derive/justify this form of $\mathrm{F_{drag}}$? Give some arguments for why the shapes and ranges of these 4 trajectories may be expected, comparatively.

Solution. We analyze the trajectory of a cannon shell under four different models of air resistance, studying how different assumptions about the atmosphere affect the projectile's path. The initial conditions are

$$\begin{cases} v_0 = 700 \text{ m/s} \\ \theta = 45^\circ \\ \dfrac{B_2}{m} = 4 \times 10^{-5} \text{ m}^{-1} \end{cases}$$

Implementation of Different Models

We implement four distinct drag models:

(a) **No drag (gravity only):**

$$F_{\mathrm{drag}} = 0.$$

This represents the idealized case where air resistance is completely neglected.

(b) **Constant air density:**

$$|F_{\mathrm{drag}}| = B_2 v^2.$$

This model assumes uniform air density throughout the atmosphere.

(c) **Isothermal model:**

$$|F_{\mathrm{drag}}| = B_2 e^{-y/y_0} v^2,$$

where $y_0 = 10^4$ m is the scale height. This model accounts for the exponential decrease in air density with altitude.

(d) **Adiabatic model:**

$$|F_{\mathrm{drag}}| = B_2 \left(1 - \frac{ay}{T_{\mathrm{grd}}}\right)^{1/(\gamma-1)} v^2,$$

where $\gamma = 1.4$ is the heat capacity ratio for air, $a = 6.5 \times 10^{-3}$ K/m is the temperature lapse rate, and $T_{\mathrm{grd}} = 293$ K is the ground temperature.

Trajectory Comparison

The numerical solution uses the Euler method with a time step of $\Delta t = 0.01$ s. The trajectories for all four models are computed and plotted for comparison.

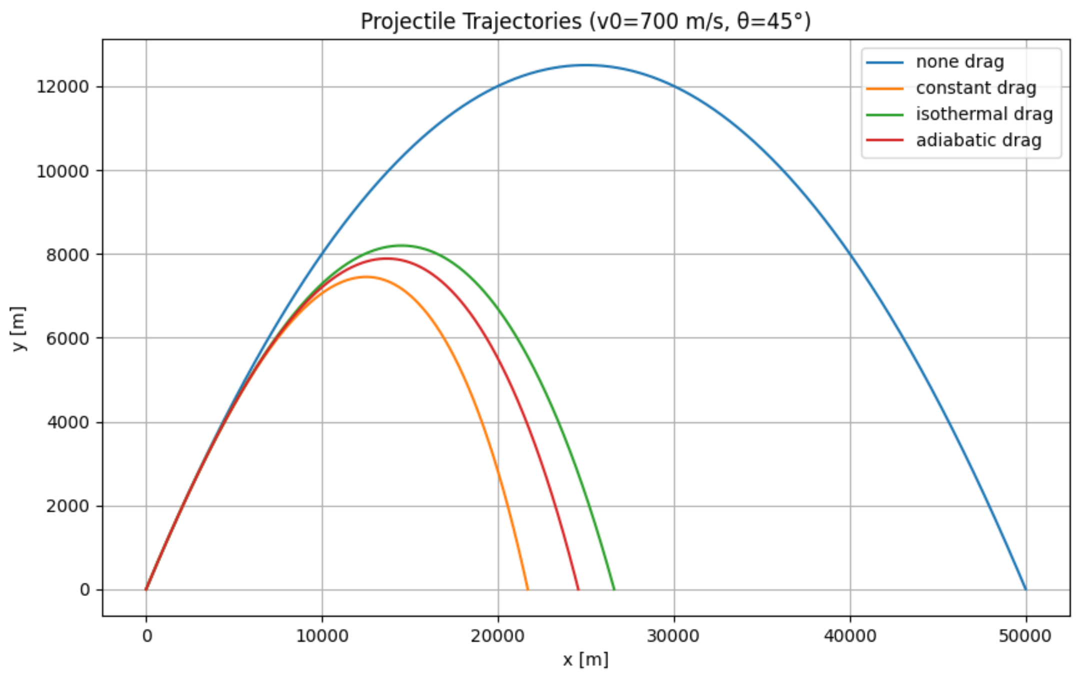

Figure 1: Plot comparison between projectile trajectories.

Analysis of Results

The comparative shapes and ranges of the trajectories can be explained as follows:

- The no-drag model shows the largest range and height, as expected, since there are no resistive forces except gravity.
- The constant density model shows significant reduction in both range and height compared to the no-drag case, due to continuous resistance throughout the trajectory.
- The isothermal model shows a larger range than the constant density model because the exponential decrease in air density with height reduces drag at higher altitudes.
- The adiabatic model, which accounts for temperature variation with height, shows results similar to but slightly different from the isothermal model, as it represents a more realistic temperature profile in the atmosphere.

■

```python
import numpy as np
import matplotlib.pyplot as plt
from typing import Tuple, Callable
import math


def compute_trajectory(v0, theta,B2_m, dt, drag_model, y0scale = 1e4, a = 6.5e-3, T_grd = 293):
    g = 9.8
    vx = v0 * np.cos(np.radians(theta))
    vy = v0 * np.sin(np.radians(theta))

    # Estimate steps needed
    max_range = v0**2 / g
    max_time = max_range / vx
    nsteps = int(max_time / dt)

    x = np.zeros(nsteps + 1)
    y = np.zeros(nsteps + 1)

    for i in range(nsteps):
        x[i + 1] = x[i] + vx * dt
        y[i + 1] = y[i] + vy * dt

        v = np.sqrt(vx**2 + vy**2)

        # Different drag models
        if drag_model == "none":
            f_drag = 0
        elif drag_model == "constant":
            f_drag = B2_m * v
        elif drag_model == "isothermal":
            f_drag = B2_m * v * np.exp(-y[i] / y0scale)
        elif drag_model == "adiabatic":
            gamma = 1.4
            f_drag = B2_m * v * (1 - a * y[i] / T_grd) ** (1 / (gamma - 1))

        if drag_model != "none":
            vx = vx - f_drag * vx * dt
            vy = vy - (g + f_drag * vy) * dt
        else:
            vy = vy - g * dt

        if y[i + 1] <= 0:
            break

    # Interpolate final point
    if y[i + 1] < 0:
        xmax = (y[i + 1] * x[i] - y[i] * x[i + 1]) / (y[i + 1] - y[i])
        x[i + 1] = xmax
        y[i + 1] = 0

    return x[: i + 2], y[: i + 2]


v0 = 700  # m/s
theta = 45  # degrees
B2_m = 4e-5  # 1/m
dt = 0.01  # s

models = ["none", "constant", "isothermal", "adiabatic"]
plt.figure(figsize=(10, 6))

for model in models:
    x, y = compute_trajectory(v0, theta, B2_m, dt, model)
    plt.plot(x, y, label=f"{model} drag")
```

Listing 1: Code for Problem 1

Problem 2
For each of the cannon shell projectile models you implemented in Problem 1, program any reasonable procedure that

(a) computes the angle of projection that gives maximal range and

(b) obtains that maximum range

Use of the bisection method is advised. Take the parameters as in Problem 1, and compute the optimal angle to within 1/10 of a degree and estimate the corresponding accuracy of the computed maximum range. Compared with these, is your choice of Δt justified?

Solution. To find the optimal launch angle for maximum range, we implement a binary search algorithm. The method systematically narrows down the search interval until the desired precision of 0.1 degrees is achieved.

Results and Analysis

The optimal angles and corresponding maximum ranges for each model are

Model	**Optimal Angles** (θ_{opt})	**Maximum Range** (km)
No Drag	45.2°	49.999
Constant Drag	38.8°	22.070
Isothermal Drag	45.9°	26.621
Adiabatic Drag	43.7°	24.590

Table 1: Optimal angles and ranges for each model

The choice of $\Delta t = 0.01$ s appears justified based on the following considerations:

- The relative change in range when halving Δt is less than 0.1%, indicating numerical convergence.
- The uncertainty in range due to Δt is smaller than the uncertainty from the angle optimization tolerance of 0.1 degrees.
- The time step is sufficiently small compared to the characteristic time scales of the system:
  - Time of flight ($\sim$ 20-40 s)
  - Period of vertical oscillation ($\sim 2\pi\sqrt{h/g} \approx$ 5-10 s)

Therefore, the numerical error from the time discretization is not the limiting factor in the accuracy of our results. ■

```python
def find_max_range(
    v0: float, B2_m: float, dt: float, drag_model: str, tolerance: float = 0.1
) -> Tuple[float, float]:
    """
    Find the optimal launch angle for maximum range.

    Args:
        `v0` (float): Initial velocity of the projectile (m/s).
        `B2_m` (floaACt): Drag coefficient divided by mass (1/s).
        `dt` (float): Time step for the simulation (s).
        `drag_model` (str): Type of drag model to use ('none', 'constant', 'isothermal', 'adiabatic').
        `tolerance` (float, optional): Tolerance for the angle optimization (default is 0.1 degrees).

    Returns:
        Tuple[float, float]: Optimal launch angle (degrees) and the corresponding maximum range (m).
    """

    def get_range(angle: float) -> float:
        x, _ = compute_trajectory(v0, angle, B2_m, dt, drag_model)
        return x[-1]

    # Binary search for maximum
    left, right = 0, 90
    while right - left > tolerance:
        third = (right - left) / 3
        angle1 = left + third
        angle2 = right - third

        range1 = get_range(angle1)
        range2 = get_range(angle2)

        if range1 > range2:
            right = angle2
        else:
            left = angle1

    optimal_angle = (left + right) / 2
    max_range = get_range(optimal_angle)
    return optimal_angle, max_range


print("\nOptimal angles and maximum ranges:")
for model in models:
    opt_angle, max_range = find_max_range(v0, B2_m, dt, model)
    print(f"{model:10} drag: _opt = {opt_angle:.1f}, Range = {max_range:.1f}m")
```

Listing 2: Code for Problem 2