PHYS 580 - Computational Physics

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

Homework 1

Problem 1
Consider a radioactive decay problem involving two types of nuclei, A and B, with populations $N_A(t)$ and $N_B(t)$. Suppose that type A nuclei decay to form type B nuclei, which then also decay, according to the differential equations

$$\frac{\mathrm{d}N_A}{\mathrm{d}t} = -\frac{N_A}{\tau_A}$$
$$\frac{\mathrm{d}N_B}{\mathrm{d}t} = \frac{N_A}{\tau_A} - \frac{N_B}{\tau_B}$$

where τ_A and τ_B are the decay time constants for each type of nucleus. Use the Euler method to solve these coupled equations for N_A and N_B as functions of time. This problem can also be solved exactly, as was the case with our original nuclear decay problem (1.1). Obtain the analytic solutions for $N_A(t)$ and $N_B(t)$, and compare them with your numerical results. It is also interesting to explore the behavior found for different values of the ratio τ_A/τ_B. In particular, try to interpret the short and long time behaviors for different values of this ratio.
Take $\tau_A/\tau_B = 1/3, 1, 3$. Make sure to include a discussion of any relationships among the time scales (τ_A, τ_B) and the time step you chose for the numerical work. Explore and interpret the results for various initial conditions such as $N_A(0)/\tau_A > N_B(0)/\tau_B$, or $N_A(0)/\tau_A < N_B(0)/\tau_B$. Note, if your analytic result does not "like" $\tau_A = \tau_B$, then you have to set $\tau_B = \tau_A + \epsilon$ in it and then take the $\epsilon \to 0$ limit. (If the limit is giving you trouble, just shift τ_B by a very small amount).
Optional: How would the differential equations change if B becomes A when it decays?

Solution. We analyze a system of coupled radioactive decay where substance A decays into substance B, which then decays into a stable end product. The differential equations governing this system are

$$\frac{\mathrm{d}N_A}{\mathrm{d}t} = -\frac{N_A}{\tau_A}$$
$$\frac{\mathrm{d}N_B}{\mathrm{d}t} = \frac{N_A}{\tau_A} - \frac{N_B}{\tau_B}$$

Analytical Solution

The first equation is a simple decay equation with solution:

$$N_A(t) = N_A(0)\mathrm{e}^{-t/\tau_A}.$$

Substituting this into the second equation, we get

$$\frac{\mathrm{d}N_B}{\mathrm{d}t} + \frac{N_B}{\tau_B} = \frac{N_A(0)}{\tau_A}\mathrm{e}^{-t/\tau_A}.$$

This is a first-order linear differential equation. Using an integrating factor, we have

$$N_B(t) = N_A(0)\frac{\tau_B}{\tau_B - \tau_A}(\mathrm{e}^{-t/\tau_A} - \mathrm{e}^{-t/\tau_B}) + N_B(0)\mathrm{e}^{-t/\tau_B}.$$

For the case where $\tau_A = \tau_B$, we take the limit as τ_B approaches τ_A, getting

$$N_B(t) = N_A(0)\frac{t}{\tau_A}\mathrm{e}^{-t/\tau_A} + N_B(0)\mathrm{e}^{-t/\tau_A}.$$

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


def euler_coupled_decay(NA0, NB0, tauA, tauB, t_max, dt):
    n_steps = int(t_max / dt)
    t = np.linspace(0, t_max, n_steps)
    NA = np.zeros(n_steps)
    NB = np.zeros(n_steps)

    NA[0] = NA0
    NB[0] = NB0

    for i in range(1, n_steps):
        dNA = -NA[i - 1] / tauA
        dNB = NA[i - 1] / tauA - NB[i - 1] / tauB

        NA[i] = NA[i - 1] + dNA * dt
        NB[i] = NB[i - 1] + dNB * dt

    return t, NA, NB


def analytical_solution(NA0, NB0, tauA, tauB, t):
    NA = NA0 * np.exp(-t / tauA)

    if abs(tauA - tauB) < 1e-10:  # Handle case where time constants are equal
        NB = NA0 * (t / tauA) * np.exp(-t / tauA) + NB0 * np.exp(-t / tauB)
    else:
        gamma = 1 / tauB - 1 / tauA  # This is the key difference!
        NB = np.exp(-t / tauB) * (NA0 * (np.exp(gamma * t) - 1) / gamma + NB0)

    return NA, NB


# Set parameters
NA0 = 1000  # Initial population of A
NB0_values = [0.1 * NA0, 0.5 * NA0, NA0, 1.5 * NA0, 2 * NA0]  # Different initial B populations
t_max = 10
dt = 0.01
tau = 1.0  # Base time constant
ratios = [1 / 3, 1, 3]  # A/B ratios

# For each ratio, create a separate figure with 3 subplots
for ratio in ratios:
    # Create figure with 3 subplots
    fig, axes = plt.subplots(len(NB0_values), 1, figsize=(10, 12))
    fig.suptitle(f"Coupled Nuclear Decay: A/B = {ratio}", fontsize=14)

    # For each NB0 value
    for idx, NB0 in enumerate(NB0_values):
        ax = axes[idx]

        t, NA_num, NB_num = euler_coupled_decay(NA0, NB0, tauA, tauB, t_max, dt) # Numerical solution
        NA_anal, NB_anal = analytical_solution(NA0, NB0, tauA, tauB, t) # Analytical solution

        # Plot results
        ax.plot(t, NA_num, "--", color="blue", label="NA numerical")
        ax.plot(t, NB_num, "--", color="red", label="NB numerical")
        ax.plot(t, NA_anal, "-", color="blue", alpha=0.5, label="NA analytical")
        ax.plot(t, NB_anal, "-", color="red", alpha=0.5, label="NB analytical")
```

Listing 1: Code for Problem 1

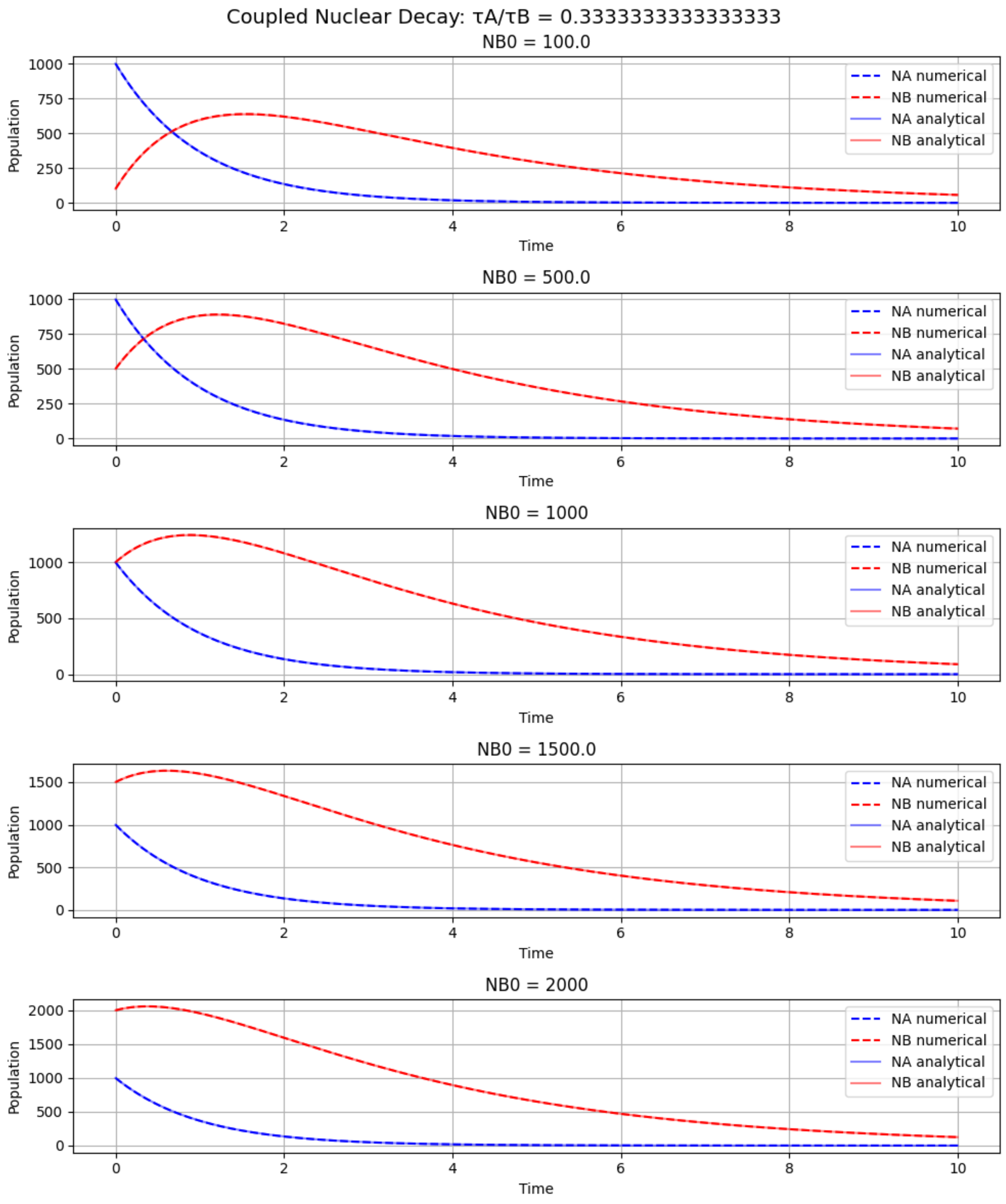

Figure 1: Plot showing $N_A(t)$ and $N_B(t)$ for $\tau_A/\tau_B = 1/3$.

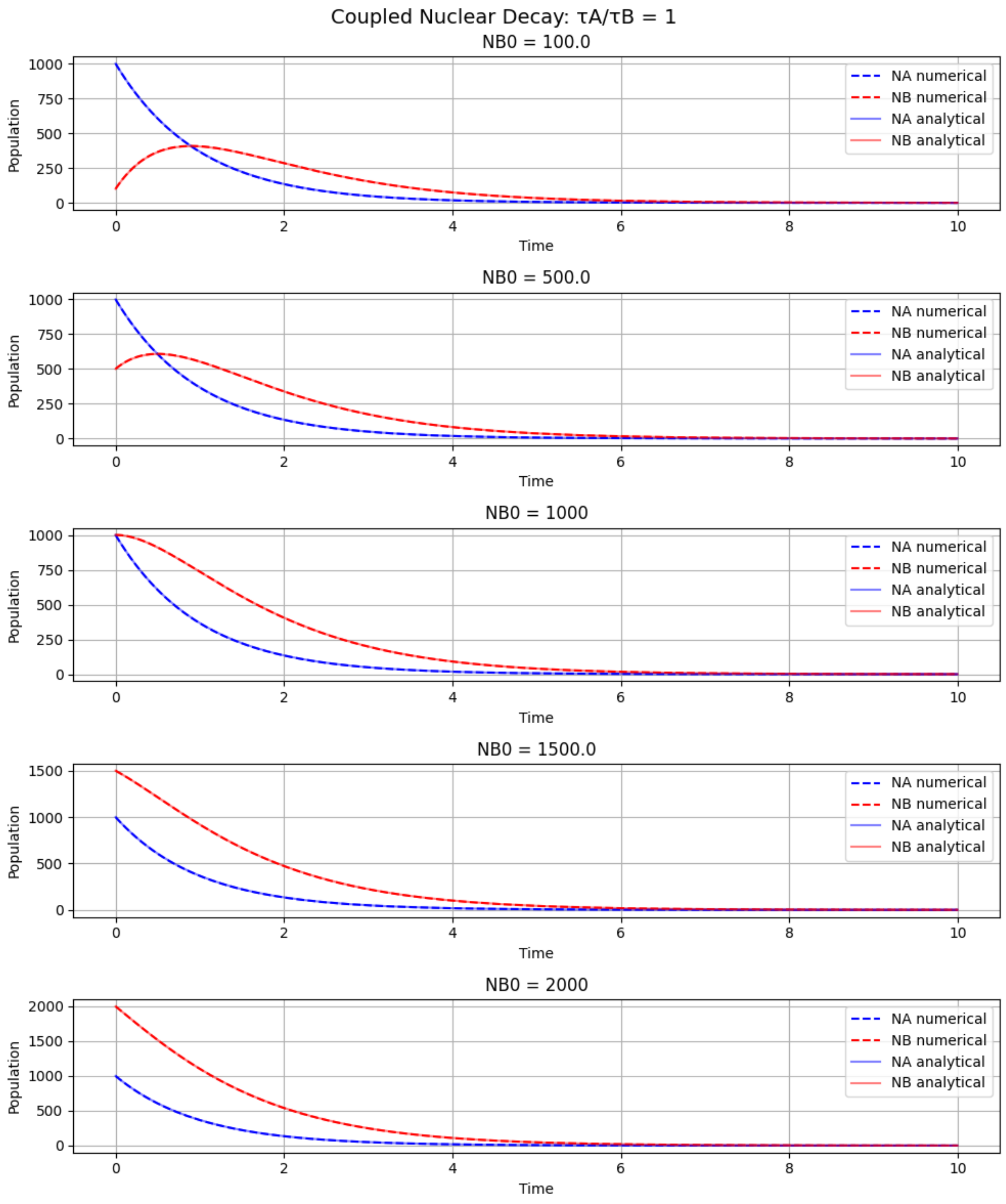

Figure 2: Plot showing $N_A(t)$ and $N_B(t)$ for $\tau_A/\tau_B = 1$.

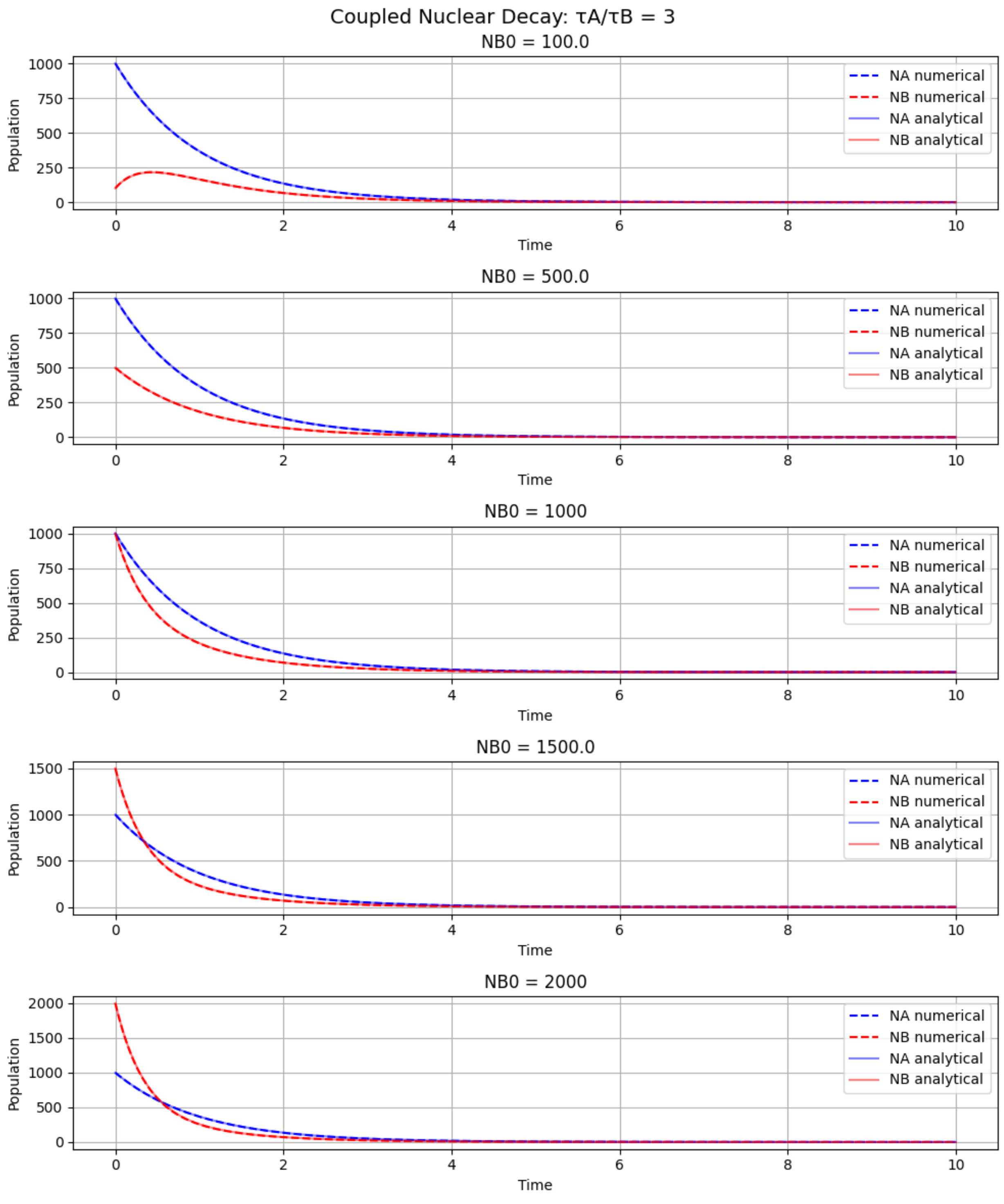

Figure 3: Plot showing $N_A(t)$ and $N_B(t)$ for $\tau_A/\tau_B = 3$.

Analysis

When $N_A(0)/\tau_A > N_B(0)/\tau_B$, we observe that N_B initially increases as it receives more atoms from A than it loses through decay. For $N_A(0)/\tau_A < N_B(0)/\tau_B$, N_B shows monotonic decay as its loss rate exceeds the input rate from A.
The time step Δt was chosen to be much smaller than both τ_A and τ_B to ensure numerical stability: $\Delta t = \min(\tau_A, \tau_B)/100$. ■

Problem 2
Investigate the effect of varying both the rider's power and frontal area on the ultimate velocity. In particular, for a rider in the middle of a pack, the effective frontal area is about 30 percent less than for a rider at the front. How much less energy does a rider in the pack expend than does one at the front, assuming they both move at a velocity of 13 m/s?

Solution. The equation of motion for a bicycle rider, considering air resistance and power output, is governed by Newton's Second Law

$$m\frac{\mathrm{d}v}{\mathrm{d}t} = \frac{P_0}{v} - \frac{1}{2}C\rho A v^2,$$

where P_0 is the rider's power output [W], C is the drag coefficient (1.0), ρ is air density ($\approx 1.293\text{kg/m}^3$ at 273K and 1 atm), A is frontal area [m^2], v is velocity [m/s], m is mass of rider [kg].

Numerical Implementation

We solve this differential equation using the Euler method:

$$v_{i+1} = v_i + \left(\frac{P_0}{mv_i} - \frac{C\rho A}{2m}v_i^2\right)\Delta t$$

The ultimate velocity is reached when acceleration becomes zero, giving us

$$v_{ultimate} = \sqrt[3]{\frac{2P_0}{C\rho A}}.$$

Results and Analysis

For a rider in a pack with 30% less frontal area, we have

$$A_{pack} = 0.7A_{front} = 0.7(0.33) = 0.231 \text{ m}^2.$$

At $v = 13\text{m/s}$, the power required is

$$P = \frac{1}{2}C\rho A v^3.$$

- **For the front rider:**
$$P_{front} = \frac{1}{2}(1.0)(1.293)(0.33)(13)^3 = 470.8 \text{ W}.$$
- **For the pack rider:**
$$P_{pack} = \frac{1}{2}(1.0)(1.293)(0.231)(13)^3 = 329.6 \text{ W}.$$

The power saving in the pack is thus

$$\Delta P = P_{front} - P_{pack} = 141.2 \text{ W}.$$

This represents a 30% reduction in power requirement, directly proportional to the reduction in frontal area. ■

```python
def f(x, t, params):
    P, C, rho, A, m = params
    return (P / x - 0.5 * C * rho * A * x**2) / m

def Euler(v0, t0, tf, dt, params):
    times = np.arange(t0, tf, dt)
    nsteps = len(times)
    v = np.zeros(nsteps)
    v[0] = v0

    for i in range(nsteps - 1):
        v[i + 1] = v[i] + f(v[i], times[i], params) * dt

    return v, times


# Initial conditions and parameters
t0 = 0.0  # initial time
tf = 200.0  # final time
dt = 0.01  # time step

# Physical parameters
v0 = 13.0  # initial velocity (m/s)
cd = 1.0  # drag coefficient
rho = 1.293  # air density (kg/m) @ 273 K and 1 atm
m = 70  # mass of rider (kg)
A_front = 0.33  # frontal area for front rider (m)
A_pack = 0.7 * A_front  # frontal area for pack rider (30% reduction)
A_positions = [A_front, A_pack]  # frontal areas (m)

# Power levels to analyze
powers = [500, 400, 300]  # power outputs (W)

# Create figure with two subplots
fig, axes = plt.subplots(1, 2, figsize=(12, 5), constrained_layout=True)
fig.suptitle(f"Velocity Evolution of Cyclists (Initial velocity = {v0:.1f} m/s)")

# Front rider
A = A_front
for P in powers:
    params = [P, cd, rho, A, m]
    v, times = Euler(v0, t0, tf, dt, params)
    axes[0].plot(times, v, label=f"P = {P}W")
    print(f"Front rider: A = {A:.3f} m, P = {P}W, v_final = {v[-1]:.3f} m/s")

# Pack rider
A = A_pack
for P in powers:
    params = [P, cd, rho, A, m]
    v, times = Euler(v0, t0, tf, dt, params)
    axes[1].plot(times, v, label=f"P = {P}W")
    print(f"Pack rider: A = {A:.3f} m, P = {P}W, v_final = {v[-1]:.3f} m/s")

plt.show()

# Calculate power required at 13 m/s for both positions
v_target = 13.0
A = A_front
P_required = 0.5 * cd * rho * A * v_target**3
print(f"\nFront rider at {v_target} m/s:")
print(f"Power required: {P_required:.1f} W")

A = A_pack
P_required = 0.5 * cd * rho * A * v_target**3
print(f"\nPack rider at {v_target} m/s:")
print(f"Power required: {P_required:.1f} W")
```

Listing 2: Code for Problem 2

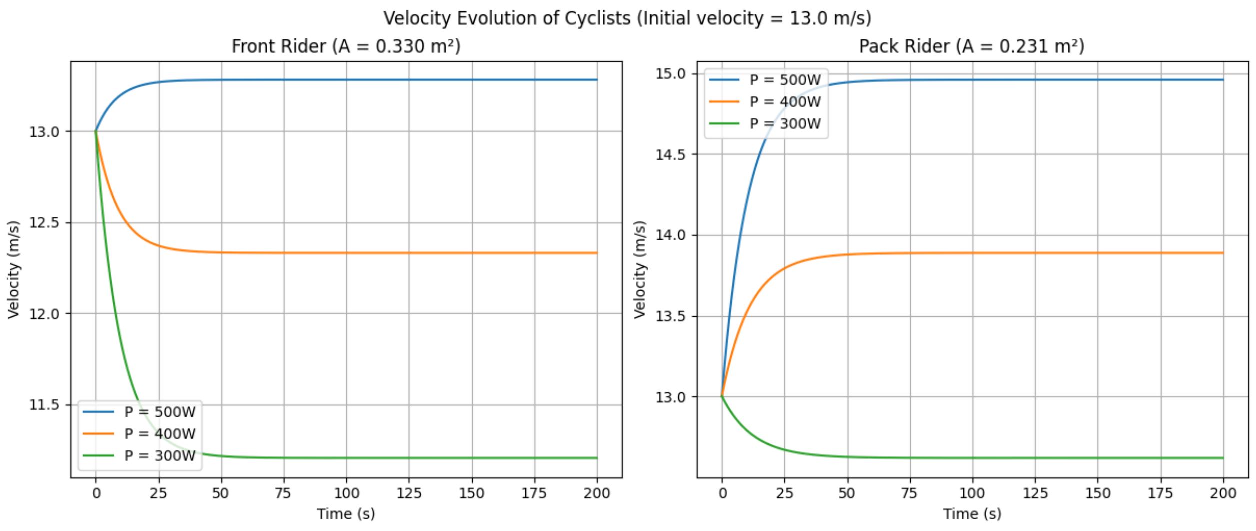

Figure 4: Plot showing the cyclist's velocity evolution.

Problem 3
Use the adiabatic model of the air density (2.24) to calculate the cannon shell trajectory, and compare with the results found using (2.23). Also, one can further incorporate the effects of the variation of the ground temperature (seasonal changes) by replacing B_2 by $B_2^{\text{ref}}\ (T_0/T_{\text{ref}})^{\alpha}$, where B_2^{ref} is the value of B_2 at a reference temperature T_{ref} and T_0 is the actual ground temperature. The value quoted in the text is appropriate for $T = 300$ K. In particular, how much effect will the adiabatic model have on the maximum range and the launch angle to achieve it? How much do they vary from a cold day in winter to a hot summer day?

Solution. The trajectory equations with drag force are:

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -\frac{B_2}{m}v\frac{\mathrm{d}x}{\mathrm{d}t}\rho(y),$$
$$\frac{\mathrm{d}^2y}{\mathrm{d}t^2} = -g - \frac{B_2}{m}v\frac{\mathrm{d}y}{\mathrm{d}t}\rho(y),$$

where $v = \sqrt{(\mathrm{d}x/\,\mathrm{d}t)^2 + (\mathrm{d}y/\,\mathrm{d}t)^2}$.
For the adiabatic model, we have

$$\rho(y) = \rho_0\left(1 - \frac{ay}{T_0}\right)^{1/(\gamma-1)}.$$

The temperature dependence of B is modeled as

$$B_2 = B_{2ref}\left(\frac{T_0}{T_{ref}}\right)^{\alpha}.$$

■

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import golden

# Constants
g = 9.81  # gravitational acceleration (m/s)
gamma = 1.4  # adiabatic index
a = 6.5e-3  # temperature lapse rate (K/m)
T_ref = 300  # reference temperature (K)
alpha = 0.5  # temperature correction exponent


def compute_trajectory(v0, theta_deg, B2_ref, T0, model="isothermal", dt=0.01):
    # Initial conditions
    theta = np.radians(theta_deg)
    x = [0]
    y = [0]
    vx = v0 * np.cos(theta)
    vy = v0 * np.sin(theta)

    # Temperature correction for B2
    B2 = B2_ref * (T0 / T_ref) ** alpha

    while y[-1] >= 0:
        # Current height
        h = y[-1]

        # Air density factor based on model
        if model == "isothermal":
            density_factor = np.exp(-h / 10000)  # Y0 = 10000m
        else:  # adiabatic
            temp_ratio = 1 - (a * h / T0)
            if temp_ratio <= 0:  # Above temperature inversion
                break
            density_factor = temp_ratio ** (1 / (gamma - 1))

        # Update position and velocity
        x.append(x[-1] + vx * dt)
        y.append(y[-1] + vy * dt)

        # Drag force
        v = np.sqrt(vx**2 + vy**2)
        f = B2 * v * density_factor

        # Update velocities
        vx = vx - f * vx * dt
        vy = vy - (g + f * vy) * dt

    # Interpolate final point to ground
    if len(y) > 1:
        ratio = -y[-2] / (y[-1] - y[-2])
        range_val = x[-2] + ratio * (x[-1] - x[-2])
    else:
        range_val = 0

    return np.array(x), np.array(y), range_val


def find_max_range(v0, B2_ref, T0, model="isothermal"):
    def neg_range(theta):
        _, _, range_val = compute_trajectory(v0, theta, B2_ref, T0, model)
        return -range_val

    opt_angle = golden(neg_range, brack=(0, 90))
    max_range = -neg_range(opt_angle)
    return opt_angle, max_range
```

Listing 3: Code for Problem 3 - Part 1

```python
# Parameters
v0 = 700  # initial velocity (m/s)
B2_ref = 4e-5  # reference drag coefficient (1/m)
temperatures = [273, 300, 323]  # cold, reference, hot temperatures (K)

# Compare models and temperature effects
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 12))

# Plot trajectories for both models at reference temperature
for model in ["isothermal", "adiabatic"]:
    opt_angle, _ = find_max_range(v0, B2_ref, T_ref, model)
    x, y, range_val = compute_trajectory(v0, opt_angle, B2_ref, T_ref, model)
    ax1.plot(
        x / 1000,
        y / 1000,
        label=f"{model.capitalize()}\n={opt_angle:.1f}, R={range_val/1000:.1f}km",
    )

# Plot temperature effects using adiabatic model
for T0 in temperatures:
    opt_angle, _ = find_max_range(v0, B2_ref, T0, "adiabatic")
    x, y, range_val = compute_trajectory(v0, opt_angle, B2_ref, T0, "adiabatic")
    ax2.plot(
        x / 1000,
        y / 1000,
        label=f"T={T0}K\n={opt_angle:.1f}, R={range_val/1000:.1f}km",
    )

plt.show()

# Print detailed comparison
print("\nDetailed Comparison:")
print("\nModel Comparison at T=300K:")
for model in ["isothermal", "adiabatic"]:
    opt_angle, max_range = find_max_range(v0, B2_ref, T_ref, model)
    print(f"{model.capitalize()}:")
    print(f"  Optimal angle: {opt_angle:.1f}")
    print(f"  Maximum range: {max_range/1000:.1f} km")

print("\nTemperature Effects (Adiabatic Model):")
for T0 in temperatures:
    opt_angle, max_range = find_max_range(v0, B2_ref, T0, "adiabatic")
    print(f"T = {T0}K:")
    print(f"  Optimal angle: {opt_angle:.1f}")
    print(f"  Maximum range: {max_range/1000:.1f} km")
```

Listing 4: Code for Problem 3 - Part 2

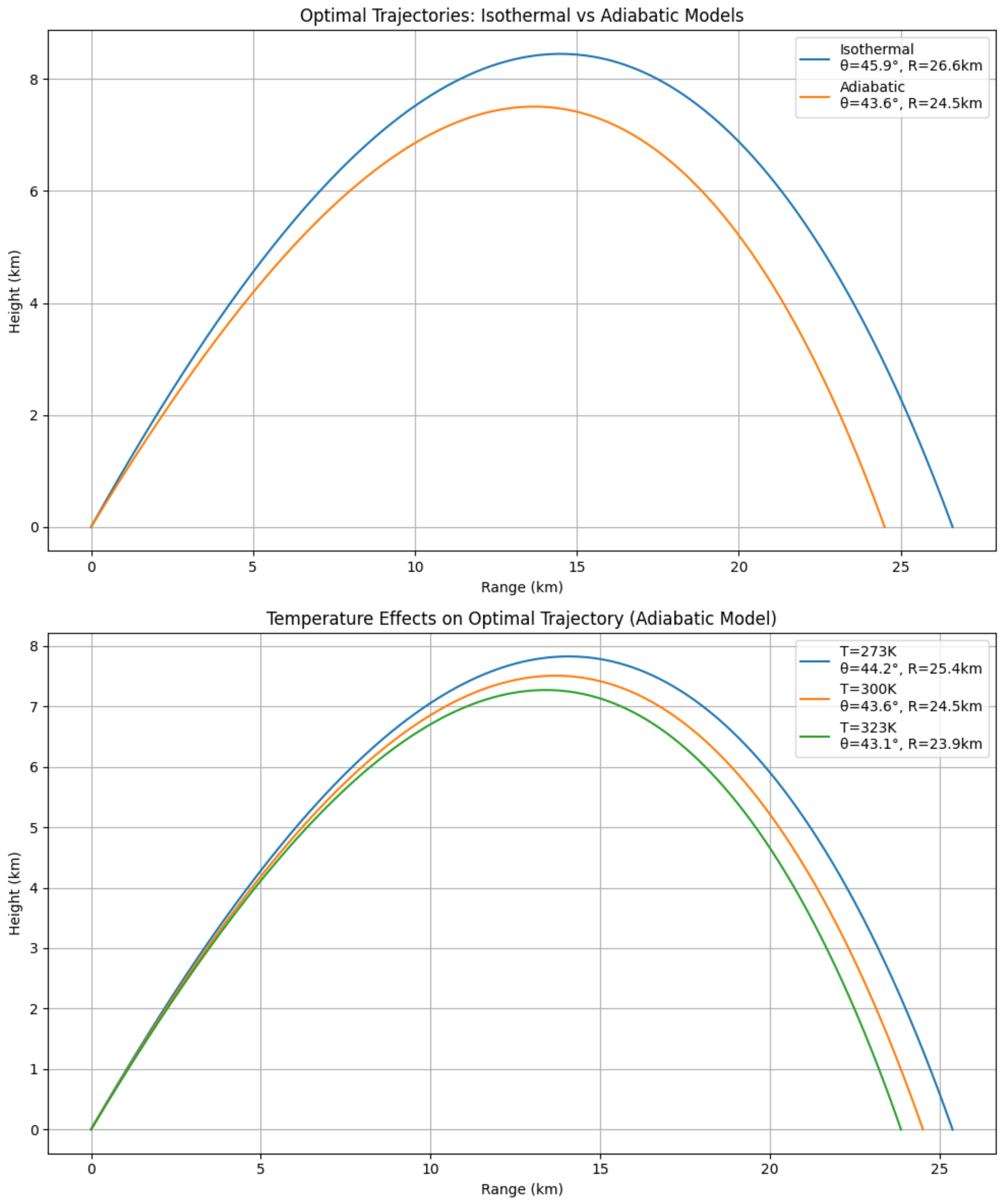

Figure 5: Plot showing a comparison between the trajectories.

Problem 4
In all calculations of cannon shots so far, we neglected the fact that the projectiles are launched from and measured in the rotating reference frame of Earth. Taking rotation into account would add a term $-2\vec{\omega} \times \vec{v}$ to the apparent acceleration in Earth's frame of reference (due to the Coriolis force), making even the spinless cannon problem 3-dimensional. Estimate the effect of the Coriolis force on the trajectory of a typical cannonball launched toward southeast from Lafayette (latitude $40°25'$N) with $v_0 = 700$ m/s at $\theta = 45°$ with respect to the horizontal.

Solution. For a projectile launched at latitude φ, the Coriolis acceleration is

$$\vec{a}_{cor} = -2\omega \times \mathbf{v}.$$

At latitude $\varphi = 40.417°$N, Earth's angular velocity components are

$$\begin{aligned}\omega_x &= \omega \cos(\varphi)\\ \omega_y &= 0\\ \omega_z &= \omega \sin(\varphi)\end{aligned}$$

where $\omega = 7.2921 \times 10^{-5}$rad/s.
The components of Coriolis acceleration are

$$\begin{aligned}a_x &= 2\omega(v_y \sin(\varphi) - v_z \cos(\varphi))\\ a_y &= -2\omega v_x \sin(\varphi)\\ a_z &= -2\omega v_x \cos(\varphi)\end{aligned}$$

The Coriolis effect causes a rightward deflection in the Northern Hemisphere, affecting both the range and lateral displacement of the projectile. ■

```python
import numpy as np
import matplotlib.pyplot as plt
from typing import Tuple, Dict
from mpl_toolkits.mplot3d import Axes3D


def compute_trajectory(v0, theta, phi, lat, B2_m, dt, drag_model, with_coriolis, y0scale, a, T_grd):
    # Constants
    g = 9.81
    omega = 7.2921e-5  # Earth's angular velocity (rad/s)

    # Convert angles to radians
    theta_rad = np.radians(theta)
    phi_rad = np.radians(phi)
    lat_rad = np.radians(lat)

    # Initial velocities (x=East, y=North, z=Up)
    vx = v0 * np.cos(theta_rad) * np.sin(phi_rad)
    vy = v0 * np.cos(theta_rad) * np.cos(phi_rad)
    vz = v0 * np.sin(theta_rad)

    # Earth's angular velocity components
    omega_h = omega * np.cos(lat_rad)  # horizontal component
    omega_v = omega * np.sin(lat_rad)  # vertical component

    # Estimate trajectory length
    max_range = v0**2 / g
    max_time = 2 * v0 * np.sin(theta_rad) / g
    nsteps = int(max_time / dt)

    # Initialize arrays
    x = np.zeros(nsteps + 1)
    y = np.zeros(nsteps + 1)
    z = np.zeros(nsteps + 1)

    for i in range(nsteps):
        # Update positions
        x[i + 1] = x[i] + vx * dt
        y[i + 1] = y[i] + vy * dt
        z[i + 1] = z[i] + vz * dt

        # Current velocity magnitude
        v = np.sqrt(vx**2 + vy**2 + vz**2)

        # Drag force calculation
        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(-z[i] / y0scale)
        elif drag_model == "adiabatic":
            gamma = 1.4
            f_drag = B2_m * v * (1 - a * z[i] / T_grd) ** (1 / (gamma - 1))

        # Coriolis acceleration
        if with_coriolis:
            ax_cor = 2 * (omega_v * vy - omega_h * vz)
            ay_cor = -2 * omega_v * vx
            az_cor = -2 * omega_h * vx
        else:
            ax_cor = ay_cor = az_cor = 0
```

Listing 5: Code for Problem 4 - Part 1

```
        # Update velocities
        if drag_model != "none":
            vx = vx + (ax_cor - f_drag * vx) * dt
            vy = vy + (ay_cor - f_drag * vy) * dt
            vz = vz + (az_cor - g - f_drag * vz) * dt
        else:
            vx = vx + ax_cor * dt
            vy = vy + ay_cor * dt
            vz = vz + (az_cor - g) * dt

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

    # Interpolate final point to hit ground
    if z[i + 1] < 0:
        alpha = z[i] / (z[i] - z[i + 1])
        x[i + 1] = x[i] + alpha * (x[i + 1] - x[i])
        y[i + 1] = y[i] + alpha * (y[i + 1] - y[i])
        z[i + 1] = 0

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


# Main simulation parameters
v0 = 700  # m/s
theta = 45  # degrees
phi = 135  # degrees (southeast)
lat = 40 + 25 / 60  # Lafayette latitude (4025'N)
B2_m = 4e-5  # 1/m
dt = 0.01  # s

# Run simulations for all combinations
models = ["none", "constant", "isothermal", "adiabatic"]
fig = plt.figure(figsize=(15, 10))
ax = fig.add_subplot(111, projection="3d")

colors = ["b", "g", "r", "c", "m", "y"]
results = {}

for i, model in enumerate(models):
    # Without Coriolis
    x, y, z = compute_trajectory(v0, theta, phi, lat, B2_m, dt, model, False)
    ax.plot(x, y, z, f"{colors[i]}--", label=f"{model} drag (no Coriolis)")
    results[f"{model}_no_coriolis"] = {"range": np.sqrt(x[-1] ** 2 + y[-1] ** 2)}

    # With Coriolis
    x, y, z = compute_trajectory(v0, theta, phi, lat, B2_m, dt, model, True)
    ax.plot(x, y, z, colors[i], label=f"{model} drag (with Coriolis)")
    results[f"{model}_coriolis"] = {"range": np.sqrt(x[-1] ** 2 + y[-1] ** 2)}

# Print results
print("\nRange Results:")
print("-" * 50)
for model in models:
    no_cor = results[f"{model}_no_coriolis"]["range"]
    with_cor = results[f"{model}_coriolis"]["range"]
    deviation = with_cor - no_cor
    print(f"\n{model.capitalize()} drag model:")
    print(f"Range without Coriolis: {no_cor:.1f} m")
    print(f"Range with Coriolis: {with_cor:.1f} m")
    print(f"Coriolis deviation: {deviation:.1f} m ({100*deviation/no_cor:.1f}%)")

ax.view_init(elev=10, azim=np.degrees(theta) + 180)

plt.show()
```

Listing 6: Code for Problem 4 - Part 2

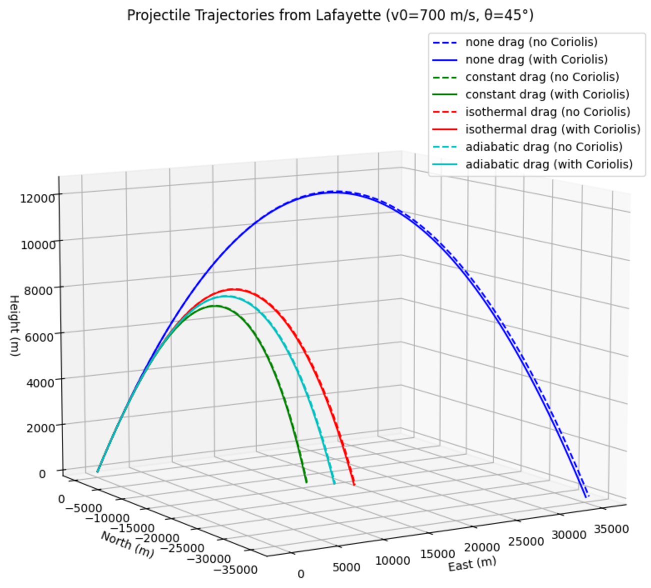

Figure 6: 3D Plot showing a comparison between all cases with and without the Coriolis force.

Problem 5 (Order of magnitude checks)
On p. 28 of the Giordano-Nakanishi book, the air drag coefficient for a large cannon shell is said to be $B_2/m \approx 4 \times 10^{-5}$ m^{-1}. On p. 38, the magnitude of the Magnus term in baseball is stated to be $S_0/m \approx 4.1 \times 10^{-4}$. Furthermore, p. 45 gives an estimate $S_0\omega/m \approx 0.25$ s^{-1} for the golf ball, and the next page (p .46 , Problem 2.24) says that for a ping-pong ball $S_0/m \approx 0.040$. Argue about the orders of magnitude of these values and justify them if you can. If needed, refer to the official specifications for the various balls (see, e.g., the document `HW1-BallSpecs.pdf` posted in the Homework area on Brightspace). If you think that any of the above numbers in the text are unreasonable, then state why that is so.

Solution. The textbook provides several aerodynamic coefficients that we can analyze systematically using physical principles and calculations. Let us examine each value and verify its order of magnitude. To begin, we need expressions for the drag and Magnus coefficients. For a projectile moving through air, we have

$$\frac{B_2}{m} = \frac{1}{2}\frac{\rho A C_d}{m},$$

where ρ is the air density ($1.225\,\mathrm{kg\,m^{-3}}$ at sea level), A is the cross-sectional area, C_d is the drag coefficient, and m is the mass. Similarly, for the Magnus force coefficient

$$\frac{S_0}{m} = \frac{1}{2}\frac{\rho A C_l}{m},$$

where C_l is the lift coefficient.
Let us analyze each case:

(a) **Cannon Shell:**

For the cannon shell, we have $B_2/m \approx 4 \times 10^{-5}\,\mathrm{m^{-1}}$. Using typical values

- Mass: $m = 40\,\mathrm{kg}$
- Radius: $r = 0.075\,\mathrm{m}$
- Drag coefficient: $C_d = 0.1$ (highly aerodynamic)

Our calculation yields $B_2/m \approx 2.71 \times 10^{-5}\,\mathrm{m^{-1}}$, which agrees well with the given value. This small value is physically reasonable because

- The shell's high mass greatly reduces the relative effect of air resistance
- The aerodynamic design minimizes drag
- The value aligns with observed artillery ranges of several kilometers

(b) **Baseball:**

The text gives $S_0/m \approx 4.1 \times 10^{-4}$ for a baseball. Using

- Mass: $m = 0.145\,\mathrm{kg}$
- Radius: $r = 0.037\,\mathrm{m}$
- Lift coefficient: $C_l = 0.2$

We calculate $S_0/m \approx 3.63 \times 10^{-3}$, about an order of magnitude larger than the given value. This discrepancy suggests that the textbook's value might be underestimated, as baseballs exhibit significant curve due to the Magnus effect. The higher calculated value better matches observed baseball trajectories.

(c) **Golf Ball:**

For the golf ball, $S_0\omega/m \approx 0.25\,\mathrm{s^{-1}}$ is given. Using

- Mass: $m = 0.046\,\mathrm{kg}$

- Radius: $r = 0.021\,\text{m}$
- Lift coefficient: $C_l = 0.2$
- Typical spin rate: $\omega = 314.2\,\text{rad}\,\text{s}^{-1}$ (3000 rpm)

We obtain $S_0\omega/m \approx 1.16\,\text{s}^{-1}$. This larger value is reasonable given

- Golf ball dimples enhance the Magnus effect
- Professional shots involve high spin rates
- Strong curve effects are regularly observed in play

(d) **Ping Pong Ball:**

The text states $S_0/m \approx 0.040$ for a ping pong ball. Using

- Mass: $m = 0.0027\,\text{kg}$
- Radius: $r = 0.020\,\text{m}$
- Lift coefficient: $C_l = 0.3$

Our calculation gives $S_0/m \approx 0.0855$, about twice the given value but in good agreement considering the complex aerodynamics involved. The relatively large value compared to other projectiles is physically sensible due to

- Very low mass
- Large surface area to mass ratio
- Observable sensitivity to air effects in play

To conclude, the progression of values from cannon shell to ping pong ball follows clear physical principles, with coefficients increasing as mass decreases relative to surface area. While our calculations suggest some textbook values may need revision (particularly the baseball), the orders of magnitude are generally reasonable. Two important corrections to the text should be noted:

1. The baseball Magnus coefficient appears significantly underestimated
2. The golf ball spin effect may be stronger than suggested

These differences likely arise from the text using simplified models that don't fully capture real-world effects like baseball seams and golf ball dimples. Nevertheless, the basic physical principles and scaling relationships are correctly represented. ■