Efficient Artillery Trajectory Approximation Method

Team 188, Problem B

November 10, 2025(UTC)

Abstract

This article focuses on the historical ballistics challenge faced by pre-modern artillery firing spherical projectiles of mass 5 kg and diameter 11 cm, with a maximum muzzle velocity of up to 450 m/s. The study aims to solve the aiming problem for targets at horizontal distances between 1,000 and 1,500 meters, accommodating complex operational conditions like varying target altitudes and wind interference, all without relying on modern computing equipment. The core methodology integrates a high-precision physical model with simplified correction coefficients for manual estimation, featuring three key modules: a dynamic International Standard Atmosphere (ISA) model to calculate altitude-dependent air density, speed of sound, and viscosity; a drag coefficient model incorporating Mach number and Reynolds number effects to accurately capture the aerodynamic drag force; and the establishment of two-dimensional and three-dimensional projectile dynamics equations to describe the trajectory under gravity and air resistance. To enable practical field use, the study derived three categories of correction coefficients—the Range Correction Factor, the Altitude Correction Factor, and the Wind Correction Factor—which quantify the necessary adjustment magnitude for the cannon's firing angle. By using pre-calculated parameter correlation tables and linear interpolation, artillery personnel can quickly optimize initial estimates for shell initial velocity and firing angle using these factors, thereby eliminating the need for complex numerical integration. This method achieves an organic combination of high-precision physical modeling and a field-usable approximation algorithm, empowering mathematically trained artillery officers to achieve precise aiming in diverse combat scenarios.

KeyWords: Artillery, Trajectory Calculation, Projectile, Exterior Ballistics

1 Introduction

1.1 Background

Artillery has a long history of development, but the 19th century marked a transition from smoothbore to rifled cannons, though spherical projectiles remained dominant in many forces until the mid-1800s. Before the advent of calculators and computers, artillery officers relied on ballistic tables and empirical formulas derived from field tests. For example, the British Army used tables compiled by William Congreve in the early 1800s, while the French developed similar systems based on the work of Gaspard Monge. These tables accounted for basic factors like range and elevation but struggled with variables such as wind, air resistance, and target altitude—limitations that often led to significant firing errors.

1.2 Problem Restatement

According to the given problem, we are supposed to confront a physical scenario where a 19th-century cannon (150–200 years old) fires a spherical projectile with a diameter of 11 cm and a mass of 5 kg, whose maximum possible muzzle velocity is 450 m/s. Naturally, we divided the problem into 2 smaller ones.

1.2.1 Problem 1

We need to determine the required muzzle velocity and launch elevation angle to successfully hit a target located 1200 m horizontally away, where both the cannon and the target are at an altitude of 500 m.

1.2.2 Problem 2

We need to address targets within a horizontal range of 1000–1500 m at varying altitudes, while accounting for the influence of wind. The core task is to develop a practical method that allows artillery officers—who possess mathematical training but no access to computers or calculators—to quickly estimate the necessary muzzle velocity and launch angle for accurate hits.

2 Notations

Symbols	Description
d	Diameter of the shell
m	Mass of the shell
A	Cross-sectional area of the shell
X	Horizontal distance between the artillery and the target
Y	Altitude/Height
g	Gravitational acceleration
ρ	Atmospheric density
T	Atmospheric temperature
μ	Dynamic viscosity of air
c_0	Local sonic speed
Re	Reynolds number
C_D	Drag coefficient
V,v	Instantaneous velocity / speed
V_x	Terminal horizontal velocity
V_y	Terminal vertical velocity
V_0	Initial velocity / speed
θ	Launch elevation
a	Acceleration
φ	Launch azimuth
M	Mach number
C_M	Compressibility effect parameter
H_M	Compressibility enhancement factor
G_M	Compressibility shift factor
ϕ	Wind azimuth

Here the main notations are defined while their specific values will be discussed and given later.

3 Assumptions

In order to simplify the artillery firing process and to get an abstract physical model without losing practical meaning, we will make some assumptions about the cannon and firing process in this subsection.

3.1 Artillery type

According to the description, target cannon is very likely a smoothbore field cannon.

First, during the mid-nineteenth century(1825–1865), smoothbore field cannons were widely used by Europeans and Americans.

Second, the spherical shape and 5 kg mass ruled out rifled cannons with elongated projectiles common in the late 19th century.

Third, the 11 cm (4.33 inch) diameter is a very close match for the standard projectile size of a 9-Pounder (approximately 4.2 inch) or a 12-Pounder (approximately 4.62 inch) smoothbore gun, like M1857 12-pounder Napoleon(Figure 3-1).

Figure 3-1: An African American soldier stands watch over a Union 12-pounder during the Civil War.[1]

According to related documents, the practical elevation range for a smoothbore field gun of the mid-19th century, was typically limited to a relatively narrow range between 0° (or slight depression) and about 5° to 8° of elevation. This low-angle trajectory was intrinsically tied to its primary military function: the smoothbore gun was designed for direct fire, meaning the gunner had to see the target to hit it. Its main roles were as an anti-personnel and anti-battery weapon, primarily using solid shot at long ranges (up to 1,500 to 1,700 yards at 5° elevation) to batter fortifications or disrupt formations, and most critically, using canister shot—a close-range, devastating shotgun-like projectile—at very low angles (approximately 0° to 1°) to break up enemy infantry or cavalry charges at distances under 400 yards. Firing at elevations significantly exceeding 8° was technically possible but increasingly impractical for a gun of this type because the spherical projectile's inaccuracy at long ranges made

aiming futile, effectively confining its utility to the flat, direct-fire roles that required minimal elevation.[2]

Therefore, in order to best fit the scenario, we would prefer trajectories with smaller firing angle (less than 10°) in our following solution to the given problem. However, we wouldn't reject other possible trajectories, for there might be cases where direct firing is impossible or people intend to test or utilize these cannons' indirect firing ability. Also, calculating indirect trajectories for cannons contributes to trajectory planning of other types of artillery as well.

3.2 Artillery firing process

We manually divide the artillery firing process into three stages.

3.2.1 Stage 1: Loading Ammunition & Adjusting Firing Angle

During Stage 1, the artillery crew loads a precisely measured charge of propellant into the breech chamber.

3.2.2 Stage 2: Ignition, Launch Dynamics and Projectile Trajectory

In Stage 2, the ignition sequence initiates rapid combustion of the propellant charge. The exothermic reaction generates high-pressure gases that accelerate the projectile through the rifled bore. This process is studied in Interior Ballistics.

Our analysis focuses on the Exterior Ballistics: the projectile's supersonic flight phase begins at muzzle exit with initial velocity V_0, encountering atmospheric drag and spin-induced Magnus effects throughout its trajectory until terminal impact.

For our theoretical framework of smoothbore field cannon, we adopt the following simplification:

1) The shell behaves as a perfectly rigid sphere body with no plastic deformation.
2) The shell is modelled as a particle when analysing its trajectory, ignoring its internal structure, spin actions and mass distribution.
3) Complex dynamics or computational fluid dynamics(CFD) are excluded.

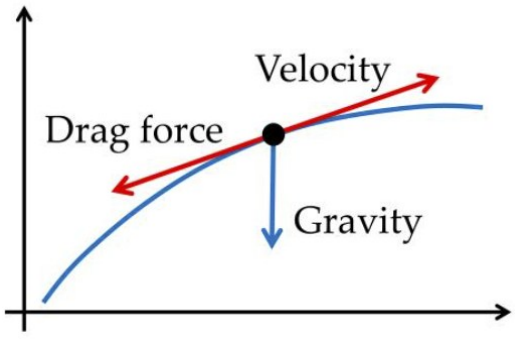

Figure 3-2: Diagram of shell dynamics

3.2.3 Stage 3: Projectile Impact.

Stage 3 defines the terminal ballistic phase where the projectile delivers its destructive effects. The lethality depends on kinetic energy transfer for solid shot, explosive fragmentation for high-explosive shells or specialized payload dispersion.

In this case, stage 3 only involves kinetic energy transfer through solid shots.

3.3 Gravity

We assume the artillery is located at 60 degrees north latitude (the UK) to fit the historical background.

Given that it's likely that the projectile won't reach higher than 10 kilometers , we assume that gravitational acceleration g=9.81m/s^2 and is constant.

4 Modelling

4.1 Free resistance model

It is an ideal trajectory model that ignores air resistance. Based on the physical laws of projectile motion, its core assumptions are that the projectile is only subject to gravity, moving in uniform linear motion in the horizontal direction and performing uniformly variable linear motion in the vertical direction.

Taking the shell as the origin, we can establish a Cartesian coordinate system, where the x-axis points toward the target and the y-axis is perpendicular to the ground, pointing up. Let the initial velocity of the projectile be V_0, and the angle between its initial velocity and the positive direction of the x-axis be θ.

$$F = m\mathrm{a} \tag{4-1}$$

According to Newton's second law (4-1), we can be derived that

$$\begin{cases} V_0 sin\theta = gt \\ x = 2V_0 cos\theta t \end{cases} \tag{4-2}$$

From the system of equations (4-2), we can obtain that

$$\theta = \frac{1}{2} arcsin(\frac{gx}{{V_0}^2}) \tag{4-3}$$

Thus, when the distance from the target is certain, the appropriate firing Angle and the initial velocity of the projectile can be obtained. For example, if x=1200m and V_0=450m/s, we can get $\theta \approx 1.67°$. Furthermore, we can obtain that when $\theta = 45°$, we have the minimum initial velocity of the shell at V_{min}=108.50m/s.

4.2 Atmospheric Properties

The trajectory simulation uses the International Standard Atmosphere (ISA) model, as implemented to calculate the local air density ρ, speed of sound c_0, and dynamic viscosity μ as a function of altitude Y.[3]

This model is critical for determining the Mach and Reynolds numbers required for the dynamic drag calculation.

4.2.1 Troposphere Model

For altitudes up to $11{,}000\mathrm{m}$ (the troposphere), the model assumes a constant temperature lapse rate $\lambda = -0.0065\ \mathrm{K/m}$. The temperature T and air density ρ are calculated using the following relationships:

$$T = T_0 + \lambda Y \tag{4-4}$$

$$\rho = \rho_0 \left(\frac{T}{T_0}\right)^{-\left(\frac{g_0}{R\lambda}\right)-1} \tag{4-5}$$

where $T_0 = 288.15\ \mathrm{K}$ is the sea level temperature, $\rho_0 = 1.225\ \mathrm{kg/m^3}$ is the sea level density, $g_0 = 9.81\ \mathrm{m/s^2}$ is the gravitational acceleration, and $R = 287.05\ \mathrm{J/(kg \cdot K)}$ is the specific gas constant for air.

4.2.2 Speed of Sound and Viscosity

The local speed of sound c_0 is derived from the temperature using the adiabatic index $\gamma = 1.4$:

$$c_0 = \sqrt{\gamma R T} \tag{4-6}$$

The dynamic viscosity μ is calculated using Sutherland's Law, which relates the viscosity to the temperature and standard sea level values:

$$\mu = \mu_0 \left(\frac{T}{T_0}\right)^{3/2} \left(\frac{T_0+S}{T+S}\right) \tag{4-7}$$

where $\mu_0 = 1.7894 \times 10^{-5}\mathrm{Pa \cdot s}$ is the sea level dynamic viscosity and $S = 110.4\ \mathrm{K}$ is Sutherland's constant. [4]

4.3 Drag Model

The projectile's dynamic drag is governed mainly by the Mach number M and the Reynolds number Re, both of which vary continuously with the projectile's velocity V and altitude Y. The drag force F_D is applied opposite to the velocity vector and is calculated using the local drag coefficient C_D:

$$F_D = \frac{1}{2}\rho V^2 A C_D \tag{4-8}$$

where A is the cross-sectional area of the projectile. The Mach number M and Reynolds number Re are defined as:

$$M = \frac{V}{c_0} \tag{4-9}$$

$$Re = \frac{\rho V d}{\mu} \tag{4-10}$$

where d is the projectile diameter.

The drag coefficient C_D model above is an improved composite model designed to accurately capture behavior across subsonic, transonic, and supersonic regimes.

However, most times it is impossible to obtain a closed-form solution for a certain object, describing the exact relation between drag coefficient C_D, instantaneous velocity V, altitude Y and other parameters.

Here, we adopt the empirical formula proposed by E. Loth et al. in their 2021 AIAA Journal paper on supersonic sphere drag coefficients (formula 7 to 8), which provides comprehensive correlations spanning Mach 0.2 to 10 and Reynolds number up to 10^6.[5]

4.3.1 Dynamic Drag Coefficient

The composite drag coefficient C_D is calculated as the sum of a low-velocity, Reynolds-dependent term and a high-velocity, Mach-dependent term:

$$C_D = \frac{24}{Re}(1 + 0.15Re^{0.687})H_M + \frac{0.42C_M}{1+(42{,}500/Re^{1.16C_M})+(G_M/Re^{0.5})} \tag{4-11}$$

Let
$$\begin{cases} Term1 = \frac{24}{Re}(1 + 0.15Re^{0.687})H_M \\ Term2 = \frac{0.42C_M}{1+(42{,}500/Re^{1.16C_M})+(G_M/Re^{0.5})} \end{cases} \quad (4\text{-}12)$$

then
$$C_D = Term1 + Term2 \quad (4\text{-}13)$$

This calculation depends on three intermediate parameters: H_M, C_M, and G_M.

4.3.2 Mach Number Functions

The function H_M accounts for the effects of compressibility on the viscous flow component, and C_M describes the Mach dependency of the pressure drag in the high-speed regime.

$$\begin{cases} H_M = \begin{cases} 0.0239M^3 + 0.212M^2 - 0.074M + 1\,, & \text{for } M < 1 \\ 0.93 + \frac{1}{3.5 + M^5}\,, & \text{for } M > 1 \end{cases} \\ C_M = \begin{cases} 1.65 + 0.65\,tanh(4M - 3.4)\,, & \text{for } M < 1.5 \\ 2.18 - 0.13\,tanh(0.9M - 2.7)\,, & \text{for } M > 1.5 \end{cases} \\ G_M = \begin{cases} 166M^3 + 3.29M^2 - 10.9M + 20\,, & \text{for } M < 0.8 \\ 5 + 40M^{-3}\,, & \text{for } M > 0.8 \end{cases} \end{cases} \quad (4\text{-}14)$$

The minimum drag coefficient is constrained such that $C_D \geq 0.01$ to ensure numerical stability and physical relevance.

4.4 Two-Dimensional Dynamic Model

The trajectory of the shell is modeled using a system of four first-order Ordinary Differential Equations (ODEs). This approach allows the numerical integration of the motion by solving for the instantaneous rates of change of the projectile's state variables.

The state vector Y is defined by the projectile's position (x, y) and its velocity components (v_x, v_y):

$$\mathbf{Y}(t) = \begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} \quad (4\text{-}15)$$

The system of governing differential equations is given by the derivative of the state vector:

$$\frac{d\mathbf{Y}}{dt} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{v}_x \\ \dot{v}_y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ a_x \\ a_y \end{pmatrix} \quad (4\text{-}16)$$

4.4.1 Acceleration Components

The total acceleration $a = (a_x, a_y)$ acting on the projectile is the sum of the accelerations due to gravity and air resistance (drag). The magnitude of the drag force F_D is defined in equation (4-8).

The acceleration due to drag is always directed opposite to the velocity vector $V = (v_x, v_y)$, where the velocity magnitude is $V = \sqrt{v_x^2 + v_y^2}$.

4.4.1.1 Horizontal Acceleration:

In the horizontal direction, the only acting force is the x-component of the drag force, $F_{D,x}$.

$$\boldsymbol{a_x = \dot{v}_x = \frac{F_{D,x}}{m} = \frac{1}{m}\left(-F_D \cdot \frac{v_x}{V}\right)} \tag{4-17}$$

Substitute the definition of F_D from (4-8) into this expression yields:

$$a_x = \frac{1}{m}\left(-\frac{1}{2}\rho V^2 A C_D \cdot \frac{v_x}{V}\right) = \left(-\frac{\rho A C_D V}{2m}\right) v_x \tag{4-18}$$

4.4.1.2 Vertical Acceleration:

In the vertical direction, the forces are gravity ($F_G = -mg$) and the y-component of the drag force ($F_{D,y}$).

$$a_y = \dot{v}_y = \frac{F_{G,y}+F_{D,y}}{m} = \frac{-mg+F_{D,y}}{m} = -g + \frac{1}{m}\left(-F_D \cdot \frac{v_y}{V}\right) \tag{4-19}$$

Similarly, substituting the expression for F_D:

$$a_y = -g + \frac{1}{m}\left(-\frac{1}{2}\rho V^2 A C_D \cdot \frac{v_y}{V}\right) = -g + \left(-\frac{\rho A C_D V}{2m}\right) v_y \tag{4-20}$$

4.4.2 System Implementation

By defining the drag acceleration factor Φ as:

$$\Phi = -\frac{\rho A C_D V}{2m} \tag{4-21}$$

The final system of ODEs used for numerical integration simplifies to:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ v_x \\ v_y \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ \Phi v_x \\ -g + \Phi v_y \end{pmatrix} \tag{4-22}$$

The numerical solver ode113 integrates this system from the initial launch conditions up to the point where the event function (defined in Section 5) indicates the projectile has returned to the initial altitude.[6]

4.5 Three-Dimensional Dynamic Model

The previously defined system of ODEs described the motion in the two-dimensional launch plane (x, y). For the simulation to address Problem 2 or accurately reflect real-world ballistics, including the influence of lateral wind components, the system must be expanded to include the third spatial dimension, z. A z-axis is defined as perpendicular to the $x-y$ launch plane, which now represent the horizonal plane, completing the right-handed Cartesian coordinate system.

The expanded state vector Y now includes the lateral position and velocity components (z, v_z):

$$\mathbf{Y}(t) = \begin{pmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{pmatrix} \tag{4-23}$$

The velocity magnitude V is updated to $V = \sqrt{v_x^2 + v_y^2 + v_z^2}$. The acceleration

vector is $a = (a_x, a_y, a_z)$, where a_x and a_y remain as previously defined in equations (4-18) and (4-20).

4.5.1 Lateral Acceleration:

In the lateral direction, the only acting force is the z-component of the drag force, $F_{D,z}$, which is directed opposite to the lateral velocity v_z.

$$a_z = \dot{v_z} = \frac{F_{D,z}}{m} = \frac{1}{m}\left(-F_D \cdot \frac{v_z}{V}\right) \tag{4-24}$$

Substituting the drag force definition from (4-8) and using the drag acceleration factor Φ from (4-21), the lateral acceleration simplifies to:

$$a_z = \left(-\frac{\rho A C_D V}{2m}\right) v_z = \Phi v_z \tag{4-25}$$

The final, comprehensive three-dimensional system of ODEs used for numerical integration is thus:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \\ \Phi v_x \\ -g + \Phi v_y \\ \Phi v_z \end{pmatrix} \tag{4-26}$$

4.5.2 Numerical Integration and Termination Conditions

4.5.2.1 Initial Conditions

The ODE system (4-26) requires a complete set of initial conditions at time $t = 0$. Assuming the projectile is launched from the origin $(0, Y_{initial}, 0)$ with a prescribed initial velocity V_0 and elevation angle θ, the initial state vector $\mathbf{Y}(0)$ is defined as:

$$\mathbf{Y}(0) = \begin{pmatrix} 0 \\ Y_{initial} \\ 0 \\ V_0 \cos\theta \\ V_0 \sin\theta \\ 0 \end{pmatrix} \tag{4-27}$$

4.5.2.2 Numerical Solver and Event Function

The six coupled first-order ODEs are numerically solved using the Adams-Bashforth-Moulton method, as implemented in MATLAB's solver “ode113”.

This method generates the projectile's state variables $(\mathbf{Y})$ at discrete time steps.

To terminate the simulation at the desired endpoint (the point of impact), an Event Function $E(t, \mathbf{Y})$ is defined. The condition for the projectile to return to the initial altitude $Y_{initial}$ is set by detecting a zero-crossing:

$$E(t, \mathbf{Y}) = y - Y_{initial} \tag{4-28}$$

The integration stops when the value of the function E crosses zero from a positive direction (i.e., when $y \to Y_{initial}$ from $y > Y_{initial}$), signaling the end of the flight path.

4.6 Wind Field Model Assumptions

To incorporate the effect of atmospheric motion on the projectile's trajectory, the following conservative assumptions are made regarding the wind field:

4.6.1 Uniform and Constant Wind Field

The wind field is modelled as a uniform and time-invariant (constant) velocity vector $\mathbf{W}$ throughout the entire flight volume. This means the wind speed and direction do not change with altitude Y, spatial position x, z, or time t during the simulation.

$$\mathbf{W} = (W_x, W_y, W_z) = \textbf{Constant} \tag{4-29}$$

4.6.2 Negligible Vertical Wind

For typical artillery trajectory simulations, the vertical component of the wind velocity is assumed to be negligible compared to the horizontal components, simplifying the model significantly:

$$\mathbf{W} = (W_x, 0, W_z) \tag{4-30}$$

The acceleration components a_x and a_z are therefore modified to account for the drag force being dependent on the relative air velocity $\boldsymbol{V_{rel}} = \boldsymbol{V} - \boldsymbol{W}$, where $\mathbf{V} = (v_x, v_y, v_z)$ is the projectile's absolute velocity vector (relative to ground). The drag acceleration factor Φ (Equation 4-21) must be calculated using the magnitude of the relative velocity $V_{rel} = \sqrt{(v_x - W_x)^2 + v_y^2 + (v_z - W_z)^2}$.

4.6.3 Horizontal Wind Azimuth

The direction of the horizontal wind is defined by the azimuth angle ϕ, which describes the orientation of the wind vector $\mathbf{W}$ in the horizontal $x - z$ plane.

ϕ is the horizontal azimuth angle of the wind vector.

$W_H = \sqrt{W_x^2 + W_z^2}$ is the magnitude of the horizontal wind speed.

The horizontal wind components are calculated from the wind speed magnitude W_H and the azimuth ϕ:

$$W_x = W_H \cdot \cos(\phi) \tag{4-31}$$

$$W_z = W_H \cdot \sin(\phi) \tag{4-32}$$

5 Model Verification & Results

5.1 Problem 1

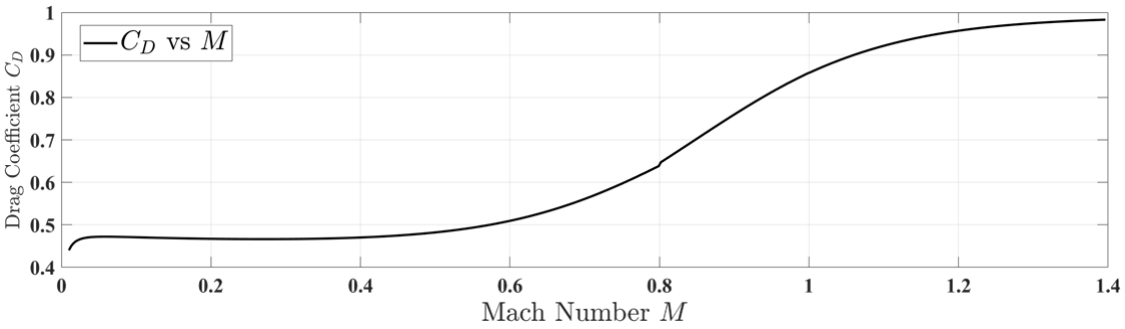

Figure 5-1: Drag Coefficient C_D vs. Mach Number M (at 500 m Altitude)

Figure 5-1 lines with the content of [Supersonic and Hypersonic Drag Coefficients for a Sphere],and this is also in line with normal circumstances where the low-speed region is dominated by the Reynolds number, while the subsonic, transonic and supersonic regions are dominated by the Mach number.

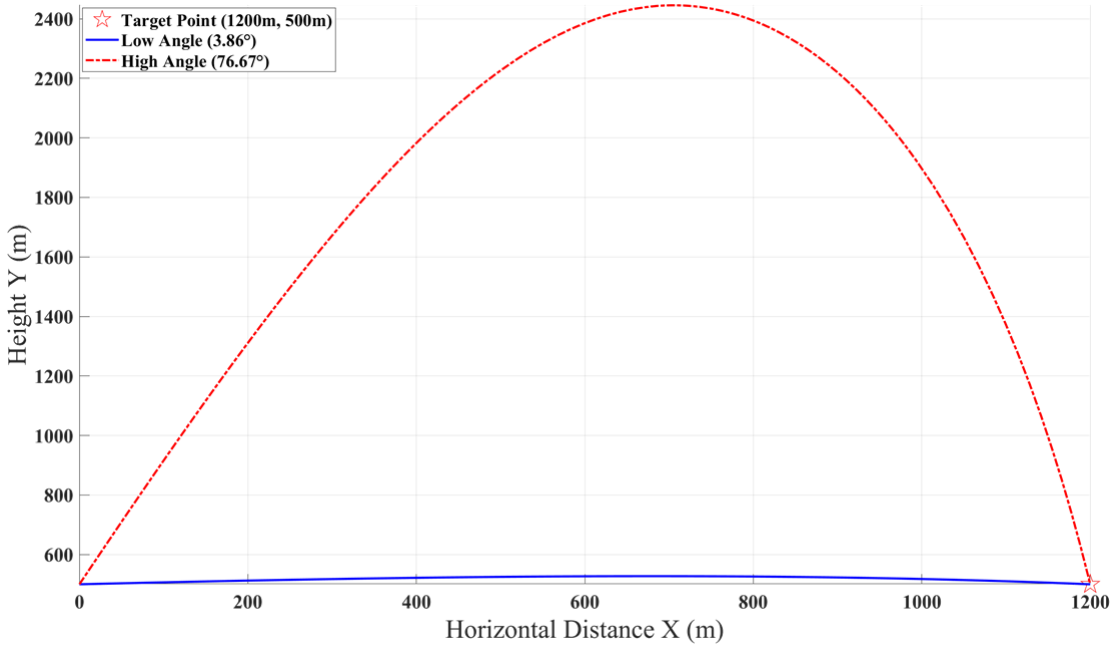

Figure 5-2: Projectile Trajectories at V_0=450.00 m/s

At the initial launch, the lateral velocity was high. Due to the effect of air resistance, the lateral velocity gradually decreased. Therefore, in the two periods before and after the shell reached its highest point, the front part of the shell was significantly farther than the rear part, which is in line with the actual situation.

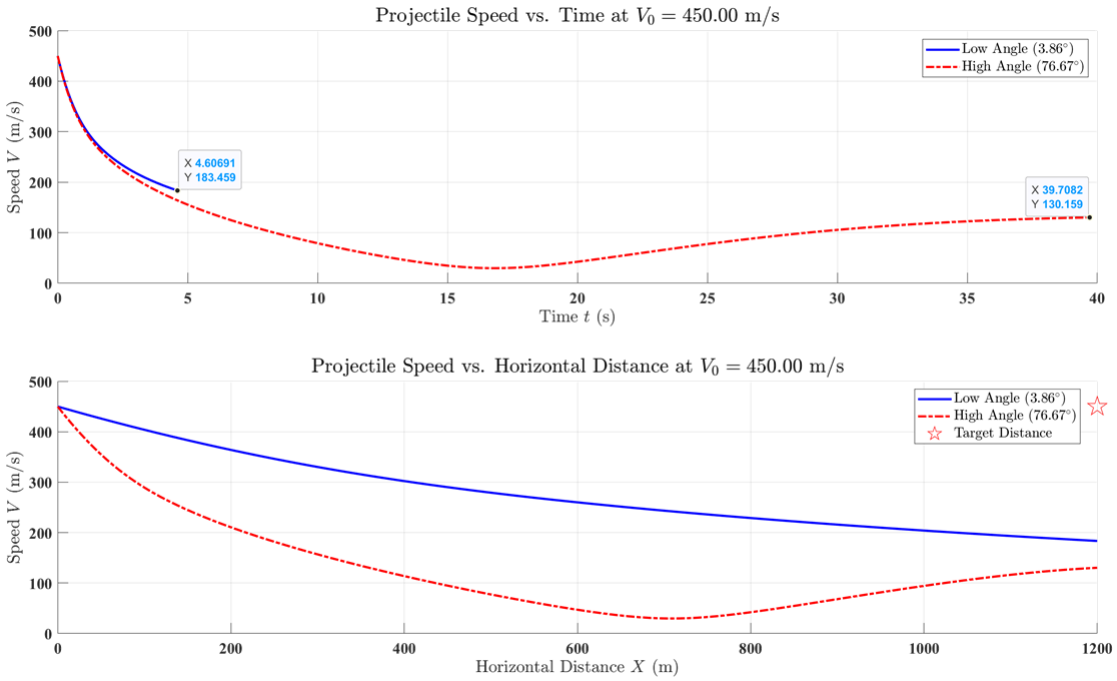

Figure 5-3: Distribution of projectile speed over time and space for θ_{min} & θ_{max}

Figure 5-3 shows the relationship curves between speed, time and distance under the same conditions. At lower angles, the gravitational force is less involved in the process and the speed is more influenced by air resistance. At a higher angle, it is clearly observable that the speed first decreases and then increases. Moreover, the decrease is much more rapid than the increase. During the speed-decreasing phase, there is no force in the vertical direction and the only resistances are air resistance and gravity. While during the speed-increasing phase, gravity does positive work and resistance does negative work, and the absolute value of their sum is smaller. Thus, the speed change is slower, which is in line with real-world experience.

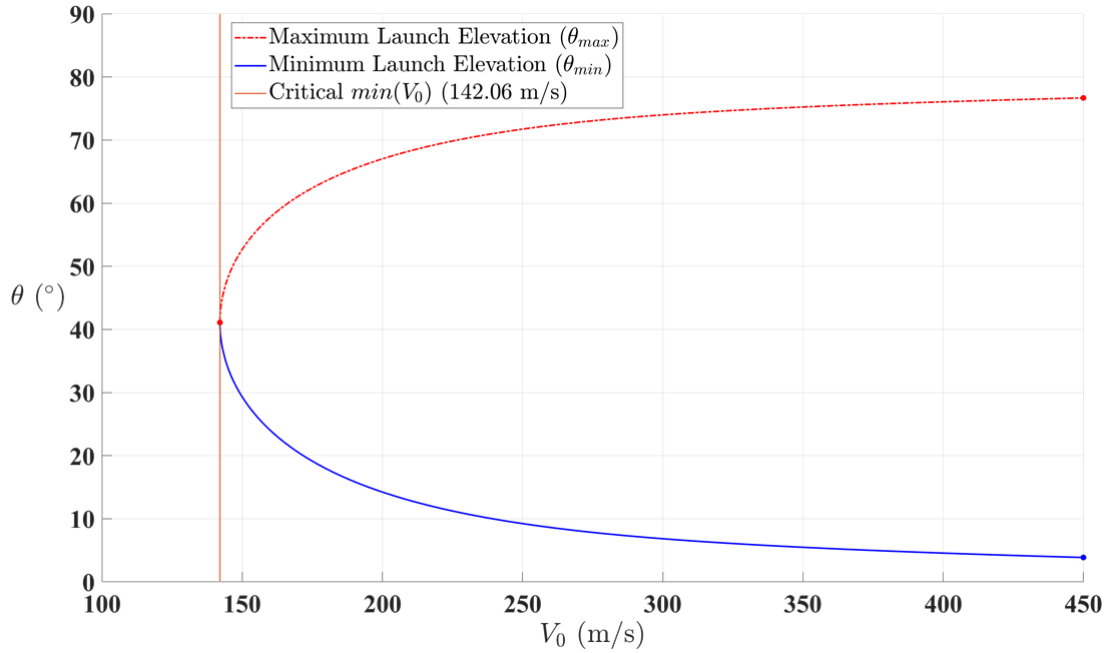

Figure 5-4: Required Launch elevation θ vs. Initial speed V_0 (X=1200m)

According to **Figure 5-4**, in the presence of air resistance, the relationship curve between the initial velocity and the launch angle has two branches (the maximum angle and the minimum angle), which matches the phenomenon we discovered from **Figure 5-2**.

Generally speaking, for any sufficiently high speed, there will be two feasible angles. Eventually, as the speed descends, two trajectories will converge into one.

At the same time, compared with free resistance model, due to energy loss contributed by drag, the minimum initial speed (142.06m/s) is greater than that in the free resistance model (108.44m/s), proving the reliability and authenticity of this model.

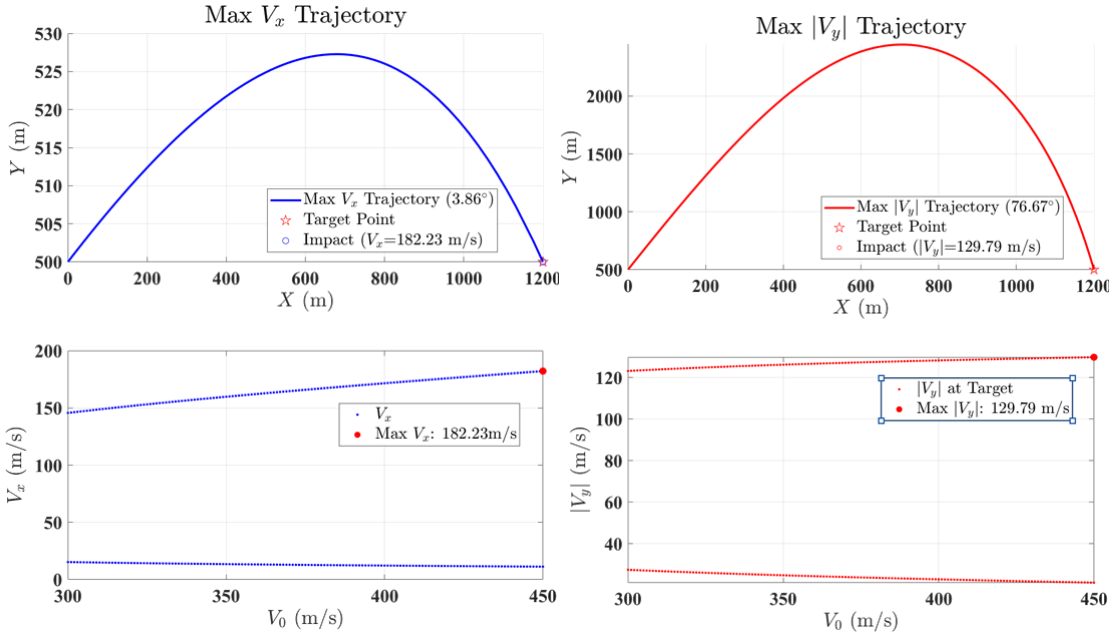

Figure 5-5: Maximum terminal speed trajectory(at V_0=450m/s) & Terminal speed vs. Initial speed

Figure 5-5 illustrates the ballistic trajectories of the projectile at a low angle of fire and a high angle of fire respectively, under the conditions of an initial velocity V_0= 450 m/s and a horizontal distance X=1200m. It has been calculated that at the low angle of fire, the maximum horizontal velocity Max V_x=182.23m/s, and at the high angle of fire, the minimum absolute value of the vertical velocity |V_y |_min= 129.79m/s.The horizontal velocity V_x reaches its maximum immediately when the projectile is launched and then decreases gradually. Similarly, the vertical velocity V_y also peaks at the moment of launch. Before the projectile reaches the highest point, V_y first decreases to 0m/s under the action of gravity and then starts to increase gradually afterward. However, the negative work done by air resistance on the projectile leads to a smaller absolute value of V_ywhen the projectile hits the target than that at the time of launch.

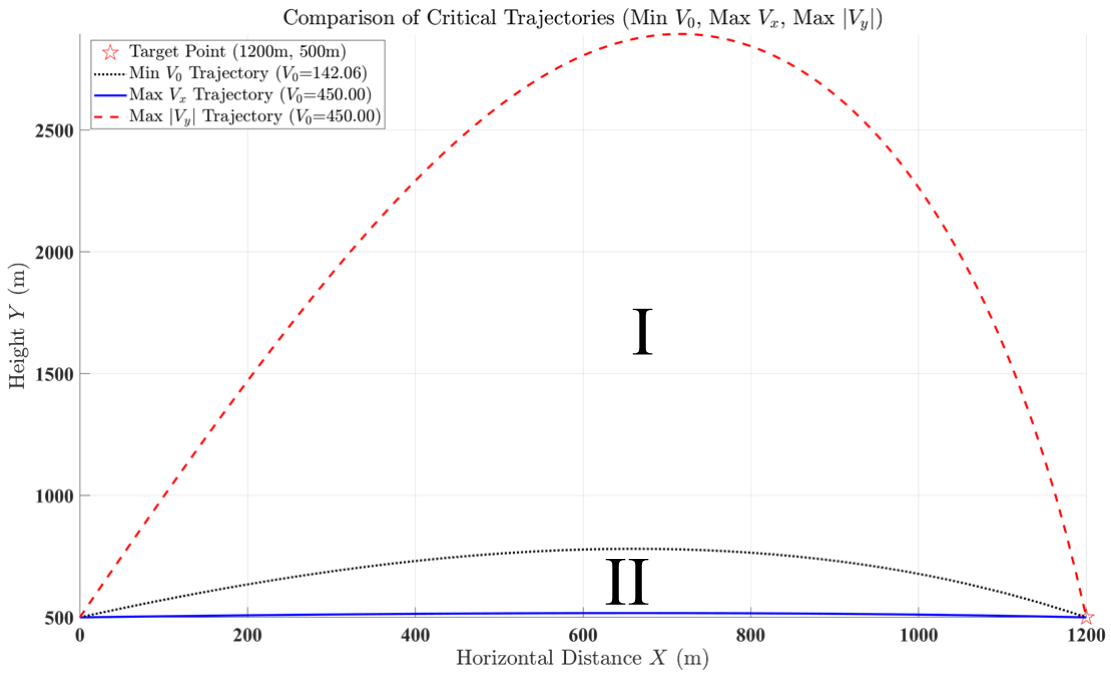

Figure 5-6: Required Launch Angle vs. Initial Velocity

Figure 5-6 is an enriched version of Figure 5-2, and the lowest feasible speed has also been included. As discussed earlier, when the speed is higher than the minimum speed, there are two trajectories. As the speed gradually decreases, there will be a trajectory that minimizes energy use by achieving the lowest speed. According to the calculation, there is only one such trajectory, and it lies between the highest initial speed trajectories, dividing the area enclosed by the two highest initial speed trajectories into section I & section II. Within each section, the closer a trajectory is to minimum V_0 trajectory, the less energy it takes to hit the target.

In conclusion, the rationality of the above figure is self-evident.

5.2 Problem 2

The development of a practical field estimation method for an artillery officer relies fundamentally on the assumption that correction factors for range (C_X), altitude (C_H), and longitudinal wind (C_W) can be treated as linear constants within the operational envelope. To validate this assumption, a linearity check was performed for C_H, C_Z, and C_W around a standard operating point ($V_0 = 400$ m/s, $X_{\text{std}} = 1200$ m, $H_{\text{std}} = 500$ m). The strength of the linear fit is assessed using the coefficient of determination (R^2).

5.2.1 Side Wind Correction Linearity

The linearity test for the lateral deviation factor, C_Z, demonstrated an exceptionally high degree of fit. This factor quantifies the lateral impact error (Z) per unit of side wind velocity (W_{side}).

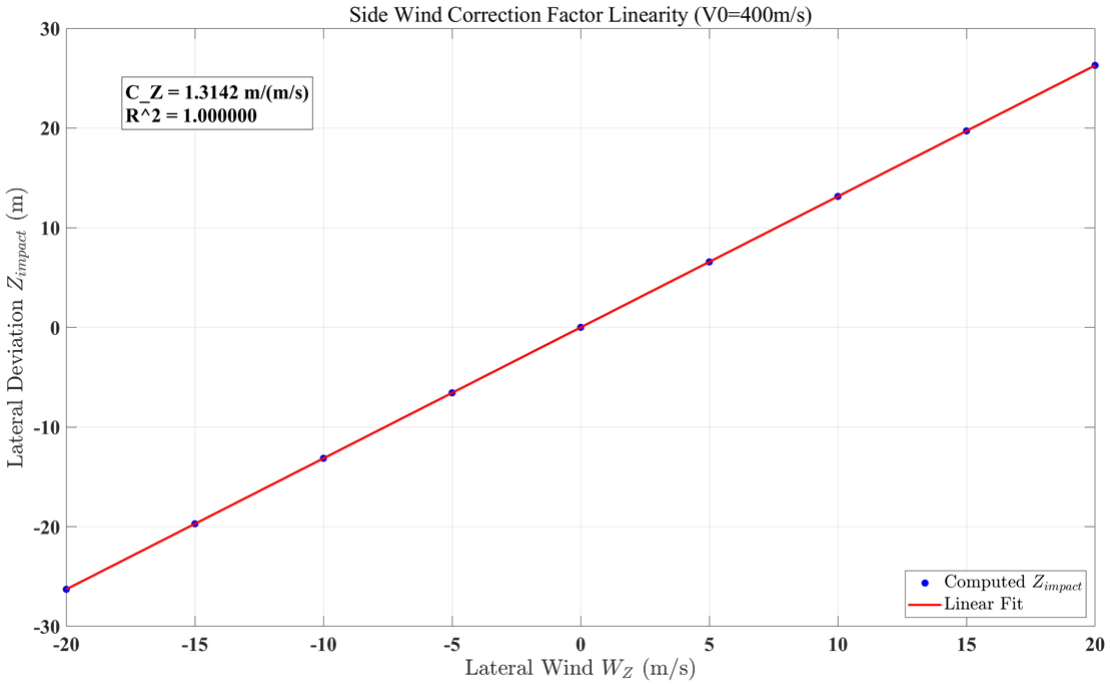

Figure 5-7 Side Wind Correction Factor Linearity (V_0=400m/s)

Linear Fit Result: $R^2 = 1.000000$

Calculated Slope (C_Z): 1.314222 m/(m/s)

The near-perfect R^2 value confirms that the side-wind correction is perfectly linear over the tested range of wind speeds. This is physically expected as the lateral drift is primarily a function of the total time of flight and the constant cross-wind force (assuming a uniform wind profile), making the lateral deviation directly proportional to the wind velocity magnitude. This validates the use of a simple, constant C_Z value for calculating lateral deflection.

5.2.2 Longitudinal Wind Correction Linearity

The longitudinal wind correction factor, C_W, is used to determine the necessary change in the firing angle ($\Delta\theta_W$) required to hit the target, given the longitudinal wind velocity (W_X).

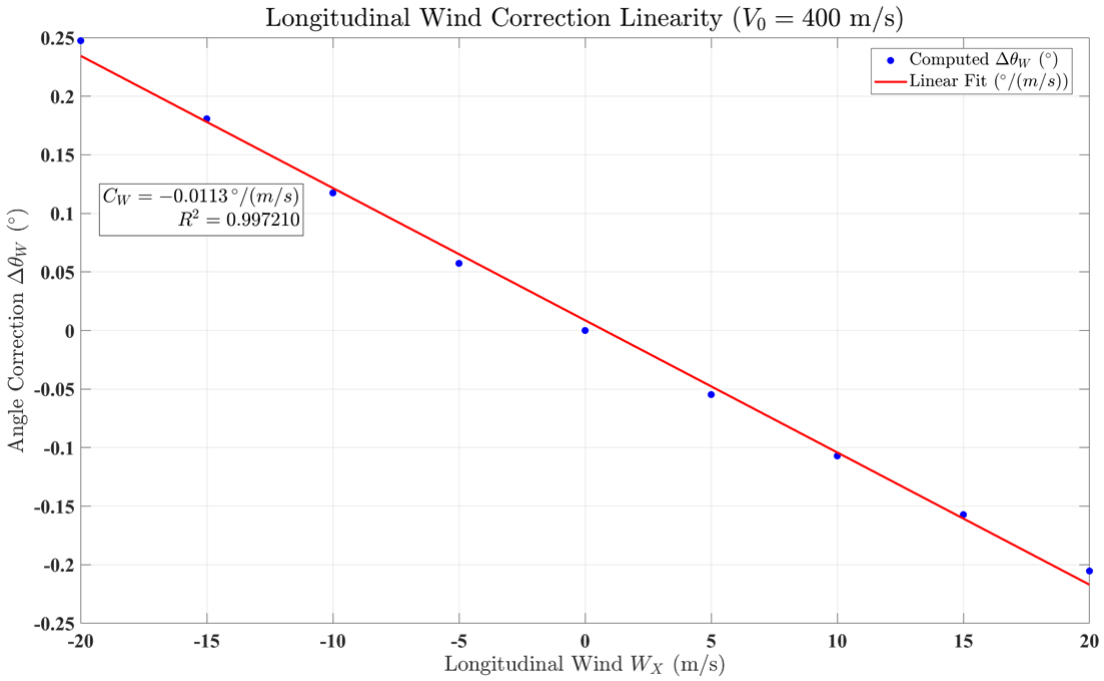

Figure 5-8 Longitudinal Wind Correction Linearity (V_0=400 m/s)

Linear Fit Result: $R^2 = 0.997210$

Calculated Slope (C_W): $-0.011287°/(\mathrm{m/s})$

The coefficient of determination is extremely close to unity, confirming that the relationship between longitudinal wind speed and the required angle correction is highly linear. The slight deviation from $R^2 = 1$ is attributable to the non-linear relationship between wind-adjusted air speed and the resulting drag force, but this effect is negligible for the purpose of field estimation. This high linearity strongly supports treating C_W as a constant factor in the officer's field calculations.

5.2.3 Altitude Correction Linearity

The altitude correction factor, C_H, accounts for the change in the required firing angle ($\Delta\theta_H$) when the actual firing altitude deviates from the standard reference altitude (ΔH). This is the most complex factor to linearize, as altitude changes significantly affect air density (ρ) and the speed of sound (c_0), thus altering the trajectory and drag profile non-linearly.

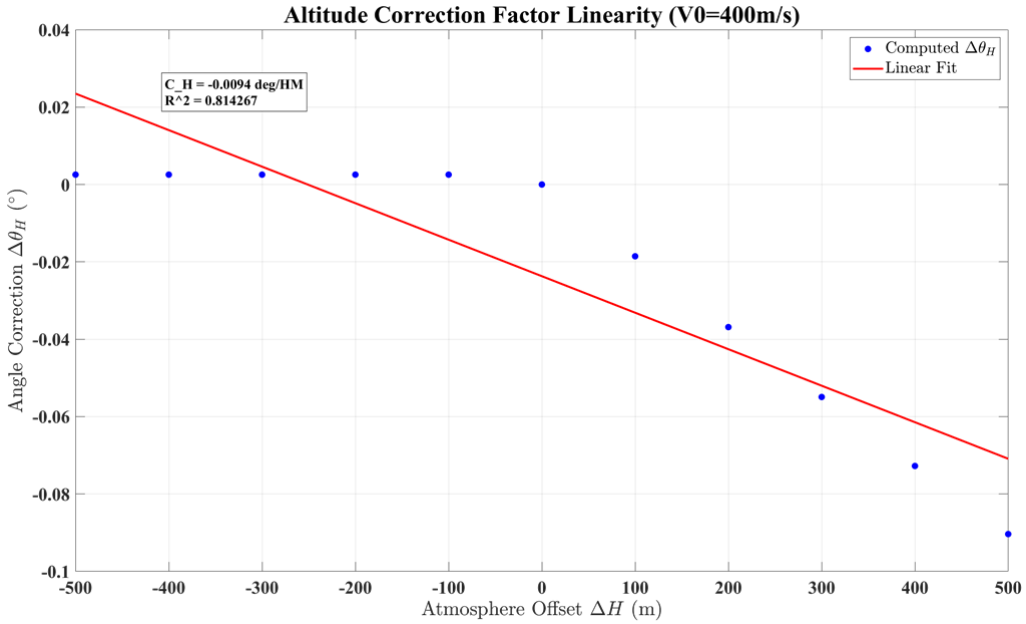

Figure 5-9 Altitude Correction Factor Linearity (V_0=400m/s)

Linear Fit Result: $R^2 = 0.814267$

Calculated Slope (C_H): $-0.0094°/(\mathrm{HM})$

While the R^2 value is the lowest of the three tests, a value above 0.8 is generally considered an acceptable fit for engineering approximations over a limited operational range. The non-linearity arises because the change in drag due to altitude is exponential, not linear. However, for a military application that prioritizes speed and simplicity without computational aids, this level of linearity is sufficient. The artillery officer can use the linear factor C_H to apply angle corrections based on observed changes in altitude, providing a quick and sufficiently accurate field correction.

6 Strength and Weaknesses

6.1 Strength

6.1.1 Model 1

Model 1 discards the assumption of a simple constant drag coefficient, instead adopting a complex nonlinear resistance model that depends on both the Mach number and the Reynolds number. This enables the model to accurately capture the severe drag variations experienced by the projectile during transonic and supersonic flight (up to 450 m/s), which is critical for this high-velocity scenario. Furthermore, the model uses high-precision numerical solvers ODE113 with stringent tolerance settings, ensuring highly accurate trajectory calculations and the robust, iterative search for the minimum critical velocity required to hit the target.

6.1.2 Model 2

The core advantage of this model lies in its balance of physical authenticity and practical operability, making it highly suitable for manual estimation by artillery officers without computer assistance. Based on the point-mass trajectory equation, it integrates the effects of air resistance, altitude's impact on air density, and 3D wind fields .It accurately calculates standard firing angles and various correction factors for different muzzle velocities. The correction factors have been verified through linearity checks to exhibit extremely strong linear correlation , enabling rapid adjustment of firing angles via simple linear calculations. Additionally, the generated range tables and correction factor tables feature a clear structure, facilitating manual lookup and interpolation. This model can fully cover the core requirements of target distances ranging from 1,000 to 1,500 meters, as well as scenarios involving different altitudes and wind conditions.

6.2 Weakness

6.2.1 Model 1

It employs a simplified 2D point-mass model, which fundamentally excludes critical real-world three-dimensional effects such as the Coriolis force, and the Magnus effect (caused by spin). Additionally, the model assumes constant environmental parameters (density, speed of sound, viscosity), neglecting the significant atmospheric variations that occur with changes in altitude and temperature over the firing range, which introduces error.

6.2.2 Model 2

The generated Excel table contains a large number of precise numerical values, which is not convenient for artillerymen without calculators to quickly look up and perform interpolation calculations. No simplified tables or charts suitable for manual use have been designed.

7 Discussion

7.1 Shape Influence

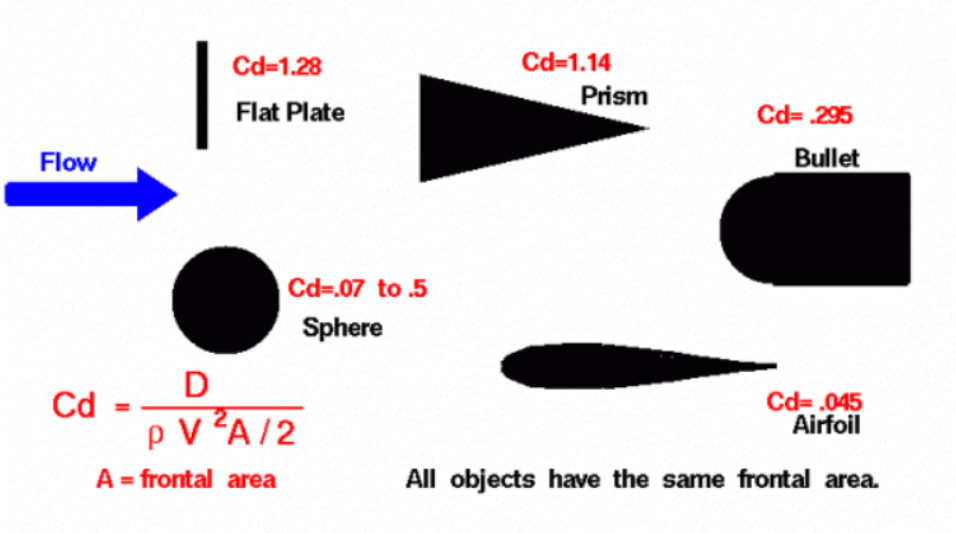

Figure7-1 Different shapes are affected by resistance[7]

After conducting research, we learned that the sphere is one of the shapes with the lowest aerodynamic efficiency. Its drag coefficient is much higher than that of the streamlined G1 projectile. Especially during subsonic flight, compared to the streamlined projectile, the drag crisis of the spherical projectile is too big to be ignored. Moreover, due to the large drag force exerted by the spherical projectile, its speed decay is much faster than that of the streamlined one, which results in a more curved trajectory for the spherical projectile. Furthermore, due to the limited historical research on the trajectory of spherical projectiles, the available reference materials are scarce, and the reference provided by the G model has become even less significant.

7.2 Calculation of air resistance on a sphere

By referring to the textbook, it is quite easy to find the general formula for the resistance coefficient.

$$C_d = \frac{1}{2} C_d \rho V^2 A \tag{7-1}$$

However, we found that the coefficient of resistance could hardly be calculated normally, because the solution for the air resistance coefficient is nonlinear.

$$\rho\left(\frac{\partial v}{\partial \tau} + (v * \nabla)v\right) = -\nabla p + \mu \nabla^2 v + f \tag{7-2}$$

When this equation is expanded, it becomes extremely complex. After that, we began to try to search for earlier experimental papers and data on the air resistance coefficient of spheres online.

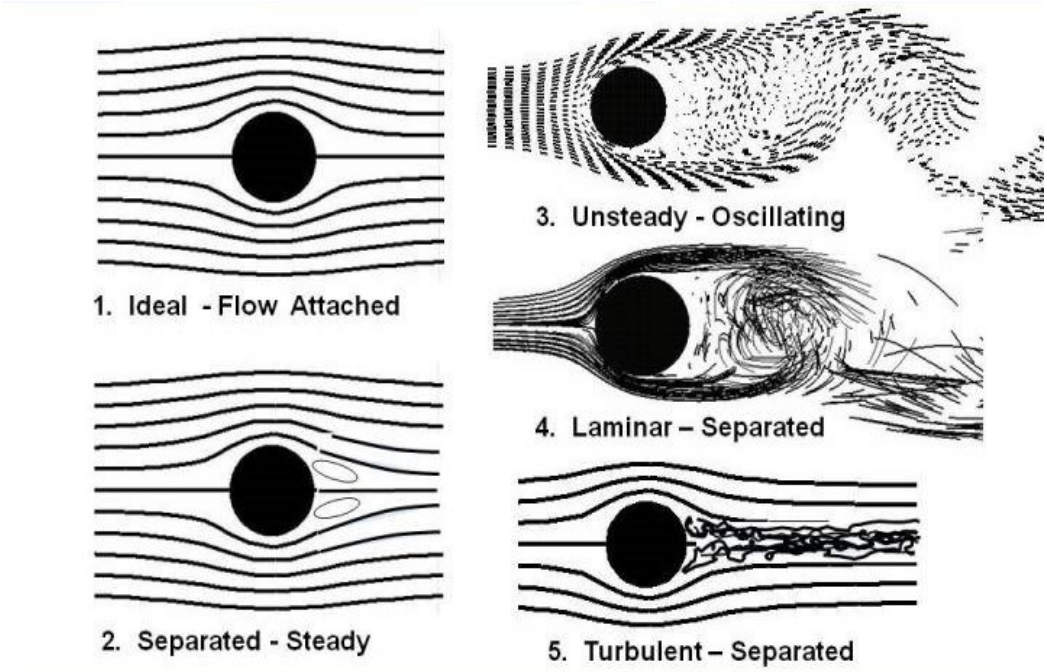

Figure7-2 Flow past a cylinder and a sphere goes through a number of transitions with velocity[8]

A tutorial on aerospace from NASA gave us some inspiration. Finally, a method using piecewise functions was found to simulate the resistance coefficient curve.

7.3 The Artillery Officer's Field Estimation Method

The objective of Problem 2 is to develop a mathematically sound, yet computationally minimal method for an artillery officer to quickly estimate firing parameters. This method relies on a pre-calculated Range Table and a Correction Factor Table, with the core operational assumption of linearity for all environmental and range deviations, as verified in Section 5.2.

The required calculations and the methodology the officer would employ are structured as follows:

7.3.1 Pre-Calculated Data Tables

The officer is provided with two essential tables (for low-angle solutions, as they offer shorter flight times and better accuracy):

7.3.1.1 Table 1: Standard Range Table

Shown in Appendix B.

(Note: $X_{std} = 1200$ m, $H_{std} = 500$ m is the reference point for all correction factors.)

7.3.2 The Estimation Procedure

The goal is to calculate the final corrected firing angle θ_{final} based on the standard angle θ_{std} and the total correction $\Delta\theta_{total}$. The officer must select a muzzle velocity V_0 and retrieve the corresponding θ_{std} and correction factors (C values) from the table.

$$\theta_{\text{final}} = \theta_{\text{std}} + \Delta\theta_{\text{total}} \tag{7-3}$$

$$\Delta\theta_{\text{total}} = \Delta\theta_X + \Delta\theta_H + \Delta\theta_W \tag{7-4}$$

7.3.2.1 Step 1: Range Correction

This corrects the angle when the actual target range (X_{target}) is different from the reference range (X_{std}).

$$\Delta\theta_X = C_X \times \left(X_{\text{target}} - X_{\text{std}}\right) \tag{7-5}$$

7.3.2.2 Step 2: Altitude Correction ($\Delta\theta_H$)

This corrects the angle for non-standard altitude of the cannon (H_{cannon}) or the target (H_{target}), where ΔH is the average change in altitude (in Hectometers, HM, where $1\ \text{HM} = 100\ \text{m}$). Since the problem primarily concerns cannon and target at the same altitude, $\Delta H = \frac{H_{cannon} - H_{std}}{100}$ is used to account for atmospheric density changes.

$$\Delta\theta_H = C_H \times \Delta H,\ \Delta H = \frac{H_{\text{actual}} - H_{\text{std}}}{100} \tag{7-6}$$

7.3.2.3 Step 3: Longitudinal Wind Correction

This corrects the angle for wind blowing along the line of fire (W_X). A positive W_X is a tailwind (decreasing θ_{final} required, hence C_W is negative) and a negative W_X is a headwind (increasing θ_{final} required).

$$\Delta\theta_W = C_W \times W_X \tag{7-7}$$

7.3.2.4 Step 4: Lateral Wind Deflection

This does not correct the firing angle but provides the necessary lateral shift to compensate for crosswind (W_{side}).

$$Z = C_Z \times W_{\text{side}} \tag{7-8}$$

The officer would then aim the cannon laterally by the distance Z into the wind to ensure the projectile lands on target. The sign of Z must be opposite to the sign of W_{side}.

7.3.3 Field Calculation Example (Mental Arithmetic)

Using $V_0 = 400\ \text{m/s}$ and targeting $X = 1300\ \text{m}$ at $H = 800\ \text{m}$ with a $5\ \text{m/s}$ headwind ($W_X = -5\ \text{m/s}$) and a $10\ \text{m/s}$ crosswind ($W_{\text{side}} = 10\ \text{m/s}$):

$$\theta_{\text{std}} = 3.55^\circ$$
$$\Delta X = 1300 - 1200 = 100\ \text{m}$$
$$\Delta\theta_X = 0.0046 \times 100 = 0.46^\circ$$
$$\Delta H = (800 - 500)/100 = 3\ \text{HM}$$
$$\Delta\theta_H = -0.0169 \times 3 \approx -0.05^\circ$$
$$W_X = -5\ \text{m/s}$$
$$\Delta\theta_W = -0.0118 \times (-5) \approx 0.06^\circ$$
$$\Delta\theta_{\text{total}} = 0.46 + (-0.05) + 0.06 = 0.47^\circ$$
$$\theta_{\text{final}} = 3.55^\circ + 0.47^\circ = 4.02^\circ$$
$$Z = 1.31 \times 10 = 13.1\ \text{m}\ \text{(Aim } 13.1\ \text{m into the wind)}.$$

This procedure transforms complex ballistic computations into simple multiplication and addition, suitable for rapid, non-digital field estimation.

8 Conclusion

Our study successfully addressed the challenge of predicting and adjusting artillery fire for a historical cannon system under varied operational conditions. The methodology relies on a high-fidelity numerical model incorporating a dynamic ISA atmosphere and a Mach-dependent drag coefficient, crucial for accurately simulating supersonic projectile flight.

For Problem 1, the numerical simulation determined that a projectile launched at a **muzzle velocity of** $400.00\ \mathrm{m/s}$ requires a **low firing angle of** $3.5453°$ to hit a target at $1{,}200\ \mathrm{m}$ distance and $500\ \mathrm{m}$ altitude.

For Problem 2, the core contribution is the development of a linear, field-ready estimation method. This method simplifies complex physics into a set of pre-calculated, constant correction factors (C_X, C_H, C_W, C_Z) that an artillery officer can use with mental arithmetic. The verification in Section 5.2 demonstrated strong linearity for the wind correction factors ($R^2 \approx 1.00$ and $R^2 \approx 0.997$) and an acceptable linear approximation for the altitude correction factor ($R^2 \approx 0.814$). This validation confirms that the proposed method is sufficiently accurate for military application where computational speed and simplicity outweigh marginal gains in precision. The resulting estimation procedure provides a robust and efficient means for adjusting for range, altitude, and wind, effectively solving the problem of field artillery adjustment in the absence of advanced computational tools.

9 References

[1] "Model 1857 12-Pounder Napoleon Field Gun," *Simple English Wikipedia, the free encyclopedia*. Jan. 06, 2025. Accessed: Nov. 09, 2025. [Online]. Available: https://simple.wikipedia.org/w/index.php?title=Model_1857_12-Pounder_Napoleon_Field_Gun&oldid=9995102

[2] "The Model 1857 12-pounder," Warfare History Network. Accessed: Nov. 09, 2025. [Online]. Available: https://warfarehistorynetwork.com/article/the-model-1857-12-pounder/

[3] *Standard Atmosphere*, ISO 2533:1975, 1975.

[4] W. Sutherland, "LII. The viscosity of gases and molecular force," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 36, no. 223, pp. 507–531, Dec. 1893, doi: 10.1080/14786449308620508.

[5] E. Loth, J. Tyler Daspit, M. Jeong, T. Nagata, and T. Nonomura, "Supersonic and Hypersonic Drag Coefficients for a Sphere," *AIAA Journal*, vol. 59, no. 8, pp. 3261–3274, Aug. 2021, doi: 10.2514/1.J060153.

[6] "ode113 - Solve nonstiff differential equations — variable order method - MATLAB." Accessed: Nov. 10, 2025. [Online]. Available: https://ww2.mathworks.cn/help/matlab/ref/ode113.html

[7] "Shape Effects on Drag," Glenn Research Center | NASA. Accessed: Nov. 09, 2025. [Online]. Available: https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/shape-effects-on-drag/

[8] "Drag of a Sphere," Glenn Research Center | NASA. Accessed: Nov. 09, 2025. [Online]. Available: https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/drag-of-a-sphere/

10 Appendices

10.1 Appendix A

Table 1

$V_0(m/s)$	$C_{x,low}$(°/m)	$C_{x,high}$(°/m)	$C_{h,low}$(°/hm)	$C_{h,high}$(°/hm)
300	0.008266686	-0.014789029	-0.027931414	0.110118096
350	0.005859174	-0.012793762	-0.021011227	0.102727611
400	0.004564003	-0.011563582	-0.016932013	0.097901745
450	0.003399761	-0.010589041	-0.013823463	0.094115775
$V_0(m/s)$	$\theta_{std,\mathrm{low}}$(°)	C_z(s)	$C_{\mathrm{w,low}}$(°/m · s^{-1})	$C_{\mathrm{w,high}}$(°/m · s^{-1})
300	6.154099057	1.656451237	-0.028433585	0.360506282
350	4.594130205	1.469976612	-0.018467797	0.355770633
400	3.545250639	1.313840864	-0.011746645	0.353406713
450	2.791072683	1.169036043	-0.007710459	0.351928825

10.2 Appendix B

Table 2 Range Table

θ(°)	V_0=300 (m/s)	V_0=350 (m/s)	V_0=400 (m/s)	V_0=450 (m/s)
1	287	374	470	573
2	522	662	809	962
3	722	898	1076	1256
4	894	1096	1295	1493
5	1046	1266	1481	1690
6	1181	1416	1641	1860
7	1301	1549	1782	2008
8	1411	1667	1908	2138
9	1511	1775	2021	2255
10	1602	1872	2122	2360
11	1685	1961	2215	2455
12	1762	2042	2299	2541
13	1833	2117	2377	2620
14	1899	2186	2448	2693
15	1960	2250	2513	2759
16	2017	2309	2573	2820
17	2069	2363	2628	2876
18	2117	2413	2679	2928
19	2162	2459	2726	2975
20	2204	2502	2770	3019
21	2242	2541	2809	3058
22	2278	2577	2845	3094
23	2310	2610	2878	3127
24	2340	2640	2908	3157

25	2367	2667	2935	3183
26	2391	2691	2959	3207
27	2414	2713	2980	3228
28	2433	2733	2999	3246
29	2451	2749	3015	3261
30	2466	2764	3029	3274
31	2479	2776	3040	3284
32	2489	2785	3048	3291
33	2498	2793	3055	3296
34	2505	2798	3059	3299
35	2509	2801	3060	3300
36	2512	2802	3060	3298
37	2512	2801	3057	3293
38	2511	2798	3052	3287
39	2507	2793	3045	3278
40	2502	2785	3036	3267
41	2495	2776	3024	3254
42	2485	2764	3011	3238
43	2474	2751	2995	3220
44	2462	2735	2977	3200
45	2447	2718	2957	3178
46	2430	2699	2935	3153
47	2412	2677	2911	3127
48	2392	2654	2885	3098
49	2370	2629	2857	3067
50	2346	2601	2827	3034
51	2320	2572	2794	2999
52	2293	2541	2760	2961
53	2263	2508	2723	2921
54	2232	2473	2685	2880
55	2199	2436	2644	2835
56	2164	2397	2601	2789
57	2128	2356	2556	2741
58	2090	2313	2510	2690
59	2050	2268	2461	2637
60	2008	2222	2410	2583
61	1964	2173	2357	2525
62	1919	2122	2301	2466
63	1871	2070	2244	2405
64	1822	2015	2185	2341
65	1772	1959	2124	2275
66	1719	1901	2061	2207
67	1665	1841	1995	2137

68	1609	1779	1928	2065
69	1551	1715	1858	1990
70	1492	1649	1787	1914
71	1431	1582	1714	1835
72	1368	1512	1638	1754
73	1304	1441	1561	1672
74	1238	1368	1482	1587
75	1170	1293	1401	1500
76	1101	1217	1318	1411
77	1030	1138	1233	1320
78	958	1059	1147	1228
79	884	977	1059	1133
80	809	894	969	1037
81	733	810	877	939
82	655	724	784	840
83	576	637	690	739
84	496	549	594	636
85	416	459	497	532
86	334	369	399	428
87	251	277	301	322
88	168	185	201	215
89	84	93	101	108

10.3 Appendix C

Table 3

Range(m)	V_0(m/s)	θ(°)	Range(m)	V_0(m/s)	θ(°)
84	300	89	2299	400	12
93	350	89	2301	400	62
101	400	89	2309	350	16
108	450	89	2310	300	23
168	300	88	2313	350	58
185	350	88	2320	300	51
201	400	88	2340	300	24
215	450	88	2341	450	64
251	300	87	2346	300	50
277	350	87	2356	350	57
287	300	1	2357	400	61
301	400	87	2360	450	10
322	450	87	2363	350	17
334	300	86	2367	300	25
369	350	86	2370	300	49
374	350	1	2377	400	13
399	400	86	2391	300	26

416	300	85	2392	300	48
428	450	86	2397	350	56
459	350	85	2405	450	63
470	400	1	2410	400	60
496	300	84	2412	300	47
497	400	85	2413	350	18
522	300	2	2414	300	27
532	450	85	2430	300	46
549	350	84	2433	300	28
573	450	1	2436	350	55
576	300	83	2447	300	45
594	400	84	2448	400	14
636	450	84	2451	300	29
637	350	83	2455	450	11
655	300	82	2459	350	19
662	350	2	2461	400	59
690	400	83	2462	300	44
722	300	3	2466	300	30
724	350	82	2466	450	62
733	300	81	2473	350	54
739	450	83	2474	300	43
784	400	82	2479	300	31
809	300	80	2485	300	42
809	400	2	2489	300	32
810	350	81	2495	300	41
840	450	82	2498	300	33
877	400	81	2502	300	40
884	300	79	2502	350	20
894	300	4	2505	300	34
894	350	80	2507	300	39
898	350	3	2508	350	53
939	450	81	2509	300	35
958	300	78	2510	400	58
962	450	2	2511	300	38
969	400	80	2512	300	36
977	350	79	2512	300	37
1030	300	77	2513	400	15
1037	450	80	2525	450	61
1046	300	5	2541	350	52
1059	350	78	2541	350	21
1059	400	79	2541	450	12
1076	400	3	2556	400	57
1096	350	4	2572	350	51
1101	300	76	2573	400	16

1133	450	79	2577	350	22
1138	350	77	2583	450	60
1147	400	78	2601	400	56
1170	300	75	2601	350	50
1181	300	6	2610	350	23
1217	350	76	2620	450	13
1228	450	78	2628	400	17
1233	400	77	2629	350	49
1238	300	74	2637	450	59
1256	450	3	2640	350	24
1266	350	5	2644	400	55
1293	350	75	2654	350	48
1295	400	4	2667	350	25
1301	300	7	2677	350	47
1304	300	73	2679	400	18
1318	400	76	2685	400	54
1320	450	77	2690	450	58
1368	350	74	2691	350	26
1368	300	72	2693	450	14
1401	400	75	2699	350	46
1411	300	8	2713	350	27
1411	450	76	2718	350	45
1416	350	6	2723	400	53
1431	300	71	2726	400	19
1441	350	73	2733	350	28
1481	400	5	2735	350	44
1482	400	74	2741	450	57
1492	300	70	2749	350	29
1493	450	4	2751	350	43
1500	450	75	2759	450	15
1511	300	9	2760	400	52
1512	350	72	2764	350	30
1549	350	7	2764	350	42
1551	300	69	2770	400	20
1561	400	73	2776	350	31
1582	350	71	2776	350	41
1587	450	74	2785	350	40
1602	300	10	2785	350	32
1609	300	68	2789	450	56
1638	400	72	2793	350	39
1641	400	6	2793	350	33
1649	350	70	2794	400	51
1665	300	67	2798	350	38
1667	350	8	2798	350	34

1672	450	73	2801	350	37
1685	300	11	2801	350	35
1690	450	5	2802	350	36
1714	400	71	2809	400	21
1715	350	69	2820	450	16
1719	300	66	2827	400	50
1754	450	72	2835	450	55
1762	300	12	2845	400	22
1772	300	65	2857	400	49
1775	350	9	2876	450	17
1779	350	68	2878	400	23
1782	400	7	2880	450	54
1787	400	70	2885	400	48
1822	300	64	2908	400	24
1833	300	13	2911	400	47
1835	450	71	2921	450	53
1841	350	67	2928	450	18
1858	400	69	2935	400	25
1860	450	6	2935	400	46
1871	300	63	2957	400	45
1872	350	10	2959	400	26
1899	300	14	2961	450	52
1901	350	66	2975	450	19
1908	400	8	2977	400	44
1914	450	70	2980	400	27
1919	300	62	2995	400	43
1928	400	68	2999	450	51
1959	350	65	2999	400	28
1960	300	15	3011	400	42
1961	350	11	3015	400	29
1964	300	61	3019	450	20
1990	450	69	3024	400	41
1995	400	67	3029	400	30
2008	450	7	3034	450	50
2008	300	60	3036	400	40
2015	350	64	3040	400	31
2017	300	16	3045	400	39
2021	400	9	3048	400	32
2042	350	12	3052	400	38
2050	300	59	3055	400	33
2061	400	66	3057	400	37
2065	450	68	3058	450	21
2069	300	17	3059	400	34
2070	350	63	3060	400	36

2090	300	58	3060	400	35
2117	350	13	3067	450	49
2117	300	18	3094	450	22
2122	400	10	3098	450	48
2122	350	62	3127	450	47
2124	400	65	3127	450	23
2128	300	57	3153	450	46
2137	450	67	3157	450	24
2138	450	8	3178	450	45
2162	300	19	3183	450	25
2164	300	56	3200	450	44
2173	350	61	3207	450	26
2185	400	64	3220	450	43
2186	350	14	3228	450	27
2199	300	55	3238	450	42
2204	300	20	3246	450	28
2207	450	66	3254	450	41
2215	400	11	3261	450	29
2222	350	60	3267	450	40
2232	300	54	3274	450	30
2242	300	21	3278	450	39
2244	400	63	3284	450	31
2250	350	15	3287	450	38
2255	450	9	3291	450	32
2263	300	53	3293	450	37
2268	350	59	3296	450	33
2275	450	65	3298	450	36
2278	300	22	3299	450	34
2293	300	52	3300	450	35

10.4 Appendix D: Trajectory_Analysis.m[1]

The code for solving problem 1.

```
%% 1. Physical Constants and Initial Conditions Definition
% Projectile Parameters
d = 0.11;
m = 5;
A = pi * (d/2)^2;
C_D_CRIT_M0 = 0.42;          % C_D,crit,M=0 (0.42), the drag crisis minimum
at Mach 0.
% Environmental & Trajectory Parameters
g = 9.81;
```

[1] Generative AI Assisted Coding on program framework, notes, variable naming, integration of separate code documents, debugging. Algorithms including iterative solution method and modelling are not included. Using Google Gemini 2.5-Flash.

```
Y_initial = 500;
X_target = 1200;
v_0 = 450;
params_base.m = m;
params_base.A = A;
params_base.g = g;
params_base.d = d;
params_base.C_D_CRIT_M0 = C_D_CRIT_M0;
params_base.Y_initial = Y_initial;
params_base.X_target = X_target;
[rho_initial, c0_initial, mu_initial] = isa_model(Y_initial);
M_max = v_0 / c0_initial;
% High-Precision Solver Setups
global ode_options fzero_options
% Define ODE options AFTER params_base is fully structured
ode_options = odeset('RelTol', 1e-9, 'AbsTol', 1e-11, 'Events', @(t, Y)
event_function(t, Y, params_base));
fzero_options = optimset('TolX', 1e-12, 'Display', 'off');
fprintf('--- Initial ISA Conditions at Y = %.0f m ---\n', Y_initial);
fprintf('Air Density (rho): %.4f kg/m^3\n', rho_initial);
fprintf('Speed of Sound (c0): %.2f m/s\n', c0_initial);
fprintf('Dynamic Viscosity (mu): %.2e Pa*s\n', mu_initial);
fprintf('Initial Max Mach Number: %.2f\n', M_max);
fprintf('-----------------------------------------------\n');
%% 2. International Standard Atmosphere (ISA) Model Function
% Calculates local atmospheric properties based on altitude Y
function [rho, c0, mu] = isa_model(Y)
    % Troposphere
    R = 287.05;          % Specific gas constant for air (J/(kg*K))
    gamma = 1.4;         % Adiabatic index (Ratio of specific heats)
    T0 = 288.15;         % Sea level temperature (K)
    rho0 = 1.225;        % Sea level density (kg/m^3)
    mu0 = 1.7894e-5;     % Sea level dynamic viscosity (Pa*s)
    S = 110.4;           % Sutherland's constant (K)
    lambda = -0.0065;    % Temperature lapse rate (K/m)
    g_0 = 9.81;          % Gravitational acceleration (m/s^2)
    Y = max(0, Y); % Altitude cannot be negative
    if Y <= 11000
    T = T0 + lambda * Y;
    rho = rho0 * (T / T0)^(-(g_0 / (R * lambda)) - 1);
    else
    % Isothermal Stratosphere (above 11km) - Simplified for this problem
scope
    T = 216.65;
```

```
    % simplification
    % For the typical ballistic range, we stay within 11km, so this is
safe.
    rho = 0.3639 * exp(-(Y-11000) * g_0 / (R * T));
    end
  c0 = sqrt(gamma * R * T);
    % Viscosity (Sutherland's Law)
  mu = mu0 * (T / T0)^(3/2) * ((T0 + S) / (T + S));
end
%% 3. Drag Coefficient C_D(M, Re) Model
function C_D = improved_C_D(v, c0, rho, d, mu, C_D_CRIT_M0)
    if v < 1e-9
        C_D = C_D_CRIT_M0;
        return;
    end
    M = v / c0;
    Re = (rho * v * d) / mu;
    % H_M calculation
    if M < 1
        H_M = 0.0239*M^3 + 0.212*M^2 - 0.074*M + 1;
    else
        H_M = 0.93 + 1/(3.5 + M^5);
    end
    % C_M calculation
    if M < 1.5
        C_M = 1.65 + 0.65*tanh(4*M - 3.4);
    else
        C_M = 2.18 - 0.13*tanh(0.9*M - 2.7);
    end
    % G_M calculation
    if M < 0.8
        G_M = 166*M^3 + 3.29*M^2 - 10.9*M + 20;
    else
        G_M = 5 + 40*(M^(-3));
    end
    % C_D calculation
    Term1 = (24/Re) * (1 + 0.15*Re^0.687) * H_M;
    Exponent = 1.16*C_M;
    Denominator = 1 + (42500/(Re^Exponent)) + (G_M/(Re^0.5));
    Term2 = (0.42 * C_M) / Denominator;
    C_D = Term1 + Term2;
    C_D = max(C_D, 0.01);
end
%% 4. Point Mass Dynamics Equation (ODE)
```

```
function dYdt = shell_dynamics(~, Y, params)
  vx = Y(3);
  vy = Y(4);
  Y_alt = Y(2); % Current height
  [rho, c0, mu] = isa_model(Y_alt);
  m = params.m;
  A = params.A;
  g = params.g;
  d = params.d;
  C_D_CRIT_M0 = params.C_D_CRIT_M0;
  v = sqrt(vx^2 + vy^2);
  if v < 1e-9
    dYdt = [vx; vy; 0; -g];
    return;
  end
  % Calculate dynamic drag coefficient
  C_D = improved_C_D(v, c0, rho, d, mu, C_D_CRIT_M0);
  drag_accel_factor = -(0.5 * rho * A * C_D * v) / m;
  ax = drag_accel_factor * vx;
  ay = -g + drag_accel_factor * vy;
  dYdt = [vx; vy; ax; ay];
end
%% 5. High-Precision Error Function and Event Handler (Used for fzero)
% --- Helper Function 1: Range Error for Root Finding (fzero) ---
function error = find_angle_error(theta, params)
  global ode_options
  % Call the trajectory solver and retrieve the final range
  [~, Y] = ode113(@(t, Y) shell_dynamics(t, Y, params), [0, 200], ...
    [0; params.Y_initial; params.v_0 * cos(theta); params.v_0 *
sin(theta)], ode_options);
  if isempty(Y)
    Range = 0;
  else
    Range = Y(end, 1);
  end
  % Error is the difference from the target distance
  error = Range - params.X_target;
end

% --- Helper Function 2: Event Function for ODE Solver ---
function [value, isterminal, direction] = event_function(~, Y, params)
  % Stops integration when the height Y(2) returns to the initial altitude
  value = Y(2) - params.Y_initial; % Height minus initial height (target
altitude)
```

```
  isterminal = 1;                % Stop integration upon event
  direction = -1;                % Event only triggered when height is
decreasing
end

% --- Helper Function 3: Negative Range for Maximization (fminbnd) ---
function neg_range = negative_range(theta, params)
  % Calculates the negative range, used for minimizing (maximizing range)
  global ode_options
  [~, Y] = ode113(@(t, Y) shell_dynamics(t, Y, params), [0, 200], ...
    [0; params.Y_initial; params.v_0 * cos(theta); params.v_0 *
sin(theta)], ode_options);
  if isempty(Y)
    neg_range = 0;
  else
    neg_range = -Y(end, 1); % Minimize -Range to maximize Range
  end
end

% --- Main Function: Find Launch Elevation Angles ---
function [theta_low_rad, theta_high_rad] = find_launch_angles(v_0,
params_base)
  global fzero_options
  params = params_base;
  params.v_0 = v_0;
  theta_low_rad = NaN;
  theta_high_rad = NaN;

  % 1. Find the actual maximum range angle (theta_max) using fminbnd
  % Minimize negative_range(theta, params) over [0.01, 89.99] degrees.
  theta_max_rad = fminbnd(@(theta) negative_range(theta, params),
deg2rad(0.01), deg2rad(89.99));
  Max_Range = -negative_range(theta_max_rad, params);
  if Max_Range < params.X_target
    return;
  end

  % 2. Search for the low angle solution in the interval [0.01 deg,
theta_max]
  try
    theta_low_rad = fzero(@(theta) find_angle_error(theta, params), ...
      [deg2rad(0.01), theta_max_rad], fzero_options);

    if abs(find_angle_error(theta_low_rad, params)) > 1e-3
```

```
      theta_low_rad = NaN;
    end
  catch
    theta_low_rad = NaN;
  end

  % 3. Search for the high angle solution in the interval [theta_max, 89.99
deg]
  try
    theta_high_rad = fzero(@(theta) find_angle_error(theta, params), ...
      [theta_max_rad, deg2rad(89.99)], fzero_options);

    if abs(find_angle_error(theta_high_rad, params)) > 1e-3
      theta_high_rad = NaN;
    end
  catch
    theta_high_rad = NaN;
  end
end

% --- Utility Function: Get Terminal State ---
function Y_end = get_terminal_state(v_0, theta, params_base)
  global ode_options
  Y_initial = params_base.Y_initial;
  Y0 = [0; Y_initial; v_0 * cos(theta); v_0 * sin(theta)];
  [~, Y] = ode113(@(t, Y) shell_dynamics(t, Y, params_base), [0, 200], Y0,
ode_options);
  if isempty(Y)
    Y_end = [NaN, NaN, NaN, NaN];
  else
    Y_end = Y(end, :);
  end
end
%% 6. Initial Angle Calculation
[theta_low_initial_rad, theta_high_initial_rad] = find_launch_angles(v_0,
params_base);
theta_low_deg = rad2deg(theta_low_initial_rad);
theta_high_deg = rad2deg(theta_high_initial_rad);
%% 7. Critical Velocity Sweep
fprintf('\n=== 7. High-Precision Iterative v_0 Sweep (Finding V_0_min)
===\n');
V_sweep = [];
Theta_low_sweep = [];
Theta_high_sweep = [];
```

```
current_v = v_0;
current_step = 1.0;
min_angle_diff_threshold = 0.01;
min_step_limit = 1e-9;
v_0_min_found = NaN;
Critical_angle_low = NaN;
Critical_angle_high = NaN;
converged = false;
stage_count = 0;
while current_step >= min_step_limit && ~converged
  stage_count = stage_count + 1;
  fprintf('-> Stage %d: step = %.10e m/s, starting v_0 = %.10e m/s\n',
stage_count, current_step, current_v);
  stage_running = true;
  last_valid_v = current_v;
  while stage_running && current_v >= 10
    [t_low, t_high] = find_launch_angles(current_v, params_base);
    if ~isnan(t_low) && ~isnan(t_high)
           % Valid solution found

      if isempty(V_sweep) || abs(V_sweep(end) - current_v) > 1e-11
        V_sweep(end+1) = current_v;
        Theta_low_sweep(end+1) = rad2deg(t_low);
        Theta_high_sweep(end+1) = rad2deg(t_high);
      end
      diff_deg = rad2deg(abs(t_high - t_low));
      if diff_deg < min_angle_diff_threshold
        fprintf('  CONVERGED: Angle diff %.2f° (< %.2f°) at v_0 = %.10e
m/s.\n', diff_deg, min_angle_diff_threshold, current_v);
        v_0_min_found = current_v;
        Critical_angle_low = rad2deg(t_low);
        Critical_angle_high = rad2deg(t_high);
        converged = true;
        stage_running = false;
        break;
      end
      last_valid_v = current_v;
      current_v = current_v - current_step;
    else
        % Overshot valid range, stop this stage and refine step
      current_v = last_valid_v;
      stage_running = false;
    end
  end
```

```
  if ~converged
    current_step = current_step * 0.1;
  end
end
if isnan(v_0_min_found) && exist('last_valid_v', 'var')
  fprintf('WARNING: Reached minimum step limit (%.1e m/s) without full
convergence. Using last valid V_0.\n', min_step_limit);
  v_0_min_found = last_valid_v;
  [t_l, t_h] = find_launch_angles(v_0_min_found, params_base);
  Critical_angle_low = rad2deg(t_l);
  Critical_angle_high = rad2deg(t_h);
end
[V_sweep, sort_idx] = sort(V_sweep);
Theta_low_sweep = Theta_low_sweep(sort_idx);
Theta_high_sweep = Theta_high_sweep(sort_idx);
fprintf('=======================================================\n');
fprintf('HIGH-PRECISION RESULT: Minimum v_0_min = %.4f m/s\n',
v_0_min_found);
if ~isnan(v_0_min_found)
  fprintf('Critical launch elevations: %.2f° and %.2f° (Diff: %.2f°)\n',
Critical_angle_low, Critical_angle_high, abs(Critical_angle_high -
Critical_angle_low));
end
fprintf('=======================================================\n');
%% 8. Critical Trajectory Solution and Data Preparation (V_0_min)
T_crit_low = []; V_crit_low = []; Y_crit_low = [];
T_crit_high = []; V_crit_high = []; Y_crit_high = [];
if ~isnan(v_0_min_found)
  Y0_crit_low = [0; Y_initial; v_0_min_found *
cos(deg2rad(Critical_angle_low)); v_0_min_found *
sin(deg2rad(Critical_angle_low))];
  [T_crit_low, Y_crit_low] = ode113(@(t, Y) shell_dynamics(t, Y,
params_base), [0, 200], Y0_crit_low, ode_options);
  V_crit_low = sqrt(Y_crit_low(:, 3).^2 + Y_crit_low(:, 4).^2);
  Y0_crit_high = [0; Y_initial; v_0_min_found *
cos(deg2rad(Critical_angle_high)); v_0_min_found *
sin(deg2rad(Critical_angle_high))];
  [T_crit_high, Y_crit_high] = ode113(@(t, Y) shell_dynamics(t, Y,
params_base), [0, 200], Y0_crit_high, ode_options);
  V_crit_high = sqrt(Y_crit_high(:, 3).^2 + Y_crit_high(:, 4).^2);
  fprintf('\n--- Trajectory Solution for v_0_min = %.2f m/s ---\n',
v_0_min_found);
  fprintf('Critical Low Angle: %.2f deg (Range: %.2f m)\n',
Critical_angle_low, Y_crit_low(end, 1));
```

```
  fprintf('Critical High Angle: %.2f deg (Range: %.2f m)\n',
Critical_angle_high, Y_crit_high(end, 1));
end
%% 9. Initial Trajectory Plotting (Figure 1)
fprintf('\n--- Trajectory Solution for Initial v_0 = %.2f m/s ---\n', v_0);
V_low = []; T_low = []; Y_low = [];
V_high = []; T_high = []; Y_high = [];
figure('Name', 'Figure 1: Trajectories and Drag Coefficient');
% ----------------- Trajectory Plot -----------------
subplot(2, 1, 1);
hold on;
plot(X_target, Y_initial, 'rp', 'MarkerSize', 10, 'DisplayName', 'Target
Point (1200m, 500m)');
if ~isnan(theta_low_deg)
  Y0_low = [0; Y_initial; v_0 * cos(theta_low_initial_rad); v_0 *
sin(theta_low_initial_rad)];
  [T_low, Y_low] = ode113(@(t, Y) shell_dynamics(t, Y, params_base), [0,
200], Y0_low, ode_options);
  V_low = sqrt(Y_low(:, 3).^2 + Y_low(:, 4).^2);
  plot(Y_low(:, 1), Y_low(:, 2), 'b-', 'LineWidth', 2, 'DisplayName',
sprintf('Low Angle (%.2f°)', theta_low_deg));
  fprintf('Low Angle (Flat Trajectory): %.2f deg (Range: %.2f m)\n',
theta_low_deg, Y_low(end, 1));
end
if ~isnan(theta_high_deg) && (isnan(theta_low_deg) || abs(theta_high_deg -
theta_low_deg) > 1)
  Y0_high = [0; Y_initial; v_0 * cos(theta_high_initial_rad); v_0 *
sin(theta_high_initial_rad)];
  [T_high, Y_high] = ode113(@(t, Y) shell_dynamics(t, Y, params_base), [0,
200], Y0_high, ode_options);
  V_high = sqrt(Y_high(:, 3).^2 + Y_high(:, 4).^2);
  plot(Y_high(:, 1), Y_high(:, 2), 'r-.', 'LineWidth', 2, 'DisplayName',
sprintf('High Angle (%.2f°)', theta_high_deg));
  fprintf('High Angle (Lobe Trajectory): %.2f deg (Range: %.2f m)\n',
theta_high_deg, Y_high(end, 1));
end
title(sprintf('Projectile Trajectories (X-Y Plane) at V_0 = %.2f m/s',
v_0));
xlabel('Horizontal Distance X (m)');
ylabel('Height Y (m)');
axis tight;
grid on;
legend('show');
% ----------------- Drag Coefficient Plot -----------------
```

```
subplot(2, 1, 2);
M_plot = linspace(0.01, M_max * 1.05, 500);
C_D_plot = arrayfun(@(m_val) improved_C_D(m_val * c0_initial, c0_initial,
rho_initial, d, mu_initial, C_D_CRIT_M0), M_plot);
plot(M_plot, C_D_plot, 'k-', 'LineWidth', 2, 'DisplayName', '$C_D$ vs
$M$');
hold on;
title(sprintf('Drag Coefficient $C_D$ vs. Mach Number
$M$ (at %.0f$\\,m$ Altitude)', Y_initial), ...
  'Interpreter', 'latex');
xlabel('Mach Number $M$', 'Interpreter', 'latex');
ylabel('Drag Coefficient $C_D$', 'Interpreter', 'latex');
grid on;
legend('Location', 'southwest', 'Interpreter', 'latex');
hold off;

%% 10. Initial Speed $V_0$ Analysis (Figure 2)
figure('Name', 'Initial Speed $V_0$ Analysis', 'NumberTitle', 'off');
% ----------------- Speed vs Time -----------------
subplot(2, 1, 1);
hold on;
if ~isempty(V_low)
  plot(T_low, V_low, 'b-', 'LineWidth', 2, ...
    'DisplayName', sprintf('Low Angle (%.2f$^{\\circ}$)', theta_low_deg));
end
if ~isempty(V_high)
  plot(T_high, V_high, 'r-.', 'LineWidth', 2, ...
    'DisplayName', sprintf('High Angle (%.2f$^{\\circ}$)',
theta_high_deg));
end
title('Projectile Speed vs. Time at $V_0 = 450.00$ m/s', 'Interpreter',
'latex');
xlabel('Time $t$ (s)', 'Interpreter', 'latex');
ylabel('Speed $V$ (m/s)', 'Interpreter', 'latex');
grid on;
legend('show', 'Location', 'northeast', 'Interpreter', 'latex');
% ----------------- Speed vs Distance -----------------
subplot(2, 1, 2);
hold on;
if ~isempty(V_low)
  plot(Y_low(:, 1), V_low, 'b-', 'LineWidth', 2, ...
    'DisplayName', sprintf('Low Angle (%.2f$^{\\circ}$)', theta_low_deg));
end
if ~isempty(V_high)
```

```
  plot(Y_high(:, 1), V_high, 'r-.', 'LineWidth', 2, ...
    'DisplayName', sprintf('High Angle (%.2f$^{\\circ}$)',
theta_high_deg));
end
plot(X_target, v_0, 'rp', 'MarkerSize', 18, 'DisplayName', 'Target
Distance');
title('Projectile Speed vs. Horizontal Distance at $V_0 = 450.00$ m/s',
'Interpreter', 'latex');
xlabel('Horizontal Distance $X$ (m)', 'Interpreter', 'latex');
ylabel('Speed $V$ (m/s)', 'Interpreter', 'latex');
grid on;
legend('show', 'Location', 'southwest', 'Interpreter', 'latex');
hold off;
%% 11. Critical Velocity Speed Analysis (Figure 3)
figure('Name', 'Figure 3: Critical Velocity Speed Analysis (V_0_min)');
if ~isnan(v_0_min_found)
% ----------------- Speed vs Time (Critical V) -----------------
  subplot(2, 1, 1);
  hold on;
  plot(T_crit_low, V_crit_low, 'b-', 'LineWidth', 2, 'DisplayName',
sprintf('Low Angle (%.2f°)', Critical_angle_low));
  plot(T_crit_high, V_crit_high, 'r-.', 'LineWidth', 2, 'DisplayName',
sprintf('High Angle (%.2f°)', Critical_angle_high));
  title(sprintf('Projectile Speed vs. Time at min(V_0)=%.2f m/s',
v_0_min_found));
  xlabel('Time t (s)');
  ylabel('Speed V (m/s)');
  grid on;
  legend('show', 'Location', 'NorthEast');
% ----------------- Speed vs Distance (Critical V) -----------------
  subplot(2, 1, 2);
  hold on;
  plot(Y_crit_low(:, 1), V_crit_low, 'b-', 'LineWidth', 2, 'DisplayName',
sprintf('Low Angle (%.2f°)', Critical_angle_low));
  plot(Y_crit_high(:, 1), V_crit_high, 'r-.', 'LineWidth', 2,
'DisplayName', sprintf('High Angle (%.2f°)', Critical_angle_high));
  plot(X_target, v_0_min_found, 'rp', 'MarkerSize', 18, 'DisplayName',
'Target Distance');
  title('Projectile Speed vs. Horizontal Distance at min(V_0)');
  xlabel('Horizontal Distance X (m)');
  ylabel('Speed V (m/s)');
  grid on;
  legend('show', 'Location', 'SouthWest');
end
```

```
hold off;
%% 12. Critical Velocity Trajectory Plot (Figure 4)
figure('Name', 'Figure 4: Critical Velocity Trajectories (V_0_min)');
hold on;
plot(X_target, Y_initial, 'rp', 'MarkerSize', 10, 'DisplayName', 'Target
Point (1200m, 500m)');
if ~isempty(Y_crit_low)
  plot(Y_crit_low(:, 1), Y_crit_low(:, 2), 'b-', 'LineWidth', 2,
'DisplayName', sprintf('Low Angle (%.2f°)', Critical_angle_low));
end
if ~isempty(Y_crit_high)
  plot(Y_crit_high(:, 1), Y_crit_high(:, 2), 'r-.', 'LineWidth', 2,
'DisplayName', sprintf('High Angle (%.2f°)', Critical_angle_high));
end
title(sprintf('Critical Trajectories at Minimum Initial Velocity
min(V_0)'));
xlabel('Horizontal Distance X (m)');
ylabel('Height Y (m)');
axis tight;
grid on;
legend('show', 'Location', 'NorthEast');
hold off;
%% 13. Angle Locus Plot (Figure 5)
figure('Name', 'Figure 5: v_0 vs. Launch Elevation Angle Locus (V_0_min)');
hold on;
if ~isempty(V_sweep)
  plot(V_sweep, Theta_high_sweep, 'r-.', 'LineWidth', 1.5, 'DisplayName',
'Maximum Launch Elevation ($\theta_{max}$)');
  plot(V_sweep, Theta_low_sweep, 'b-', 'LineWidth', 1.5, 'DisplayName',
'Minimum Launch Elevation ($\theta_{min}$)');
  idx_450 = find(V_sweep == v_0, 1);
  if ~isempty(idx_450)
    scatter(V_sweep(idx_450), Theta_low_sweep(idx_450), 30, 'b', 'filled',
'HandleVisibility', 'off');
    scatter(V_sweep(idx_450), Theta_high_sweep(idx_450), 30, 'r', 'filled',
'HandleVisibility', 'off');
  end
  if ~isnan(v_0_min_found)
    plot([v_0_min_found, v_0_min_found], [0, 90], 'Color', [0.85 0.33 0.1],
'LineStyle', '-', 'LineWidth', 1, 'DisplayName', sprintf('Critical
$min(V_0)$ (%.2f m/s)', v_0_min_found));
    scatter(v_0_min_found, Critical_angle_low, 30, 'b', 'filled',
'HandleVisibility', 'off');
```

```
    scatter(v_0_min_found, Critical_angle_high, 30, 'r', 'filled',
'HandleVisibility', 'off');
  end
end
title(sprintf('Required Launch Elevation vs. Initial Velocity (Target: %.0f
m)', X_target), 'Interpreter', 'latex');
xlabel('Initial Speed $V_0$ (m/s)', 'Interpreter', 'latex');
ylabel('Launch Elevation $\theta$ ($^\circ$)', 'Interpreter', 'latex');
grid on;
legend('show', 'Location', 'NorthWest', 'Interpreter', 'latex');
hold off;
%% 14. Maximize Terminal Speed Sweep
function [max_v, best_v0, best_theta_deg, V_log, V_terminal_log] =
sweep_for_max_terminal_speed(params_base, target_v_index, v_start, v_end,
step_coarse, step_fine)
% target_v_index: 3 for Vx (horizontal), 4 for Vy (vertical)
  V_log = [];
  V_terminal_log = [];
  max_v = -Inf;
  best_v0 = NaN;
  best_theta_deg = NaN;
  v_sweep_coarse = v_start:step_coarse:v_end;
  for v_0_test = v_sweep_coarse
    [t_low, t_high] = find_launch_angles(v_0_test, params_base);
    angles = [t_low, t_high];
    for theta = angles
      if ~isnan(theta)
        Y_end = get_terminal_state(v_0_test, theta, params_base);
        v_terminal = abs(Y_end(target_v_index));
        V_log(end+1) = v_0_test;
        V_terminal_log(end+1) = v_terminal;
        if v_terminal > max_v
          max_v = v_terminal;
          best_v0 = v_0_test;
          best_theta_deg = rad2deg(theta);
        end
      end
    end
  end
% Fine Sweep around the best coarse result
  if ~isnan(best_v0)
    v_start_fine = max(v_start, best_v0 - step_coarse);
    v_end_fine = min(v_end, best_v0 + step_coarse);
    v_sweep_fine = v_start_fine:step_fine:v_end_fine;
```

```
    for v_0_test = v_sweep_fine
      [t_low, t_high] = find_launch_angles(v_0_test, params_base);
      angles = [t_low, t_high];
      for theta = angles
        if ~isnan(theta)
          Y_end = get_terminal_state(v_0_test, theta, params_base);
          v_terminal = abs(Y_end(target_v_index));
          if v_terminal > max_v
            max_v = v_terminal;
            best_v0 = v_0_test;
            best_theta_deg = rad2deg(theta);
          end
        end
      end
    end
  end
end
% --- Run Max Vx Optimization (Index 3) ---
V_start_opt = 300;
V_end_opt = 450;
Step_coarse = 1;
Step_fine = 0.1;
fprintf('\n=== 15. Maximize Terminal Horizontal Speed (Vx) Sweep ===\n');
[Max_Vx, Best_V0_Vx, Best_Theta_Vx, V_log_Vx, V_terminal_log_Vx] =
sweep_for_max_terminal_speed(params_base, 3, V_start_opt, V_end_opt,
Step_coarse, Step_fine);
Max_Kx = 0.5 * m * Max_Vx^2;
fprintf('MAX VX RESULT:\n');
fprintf('Max Terminal Vx: %.4f m/s\n', Max_Vx);
fprintf('Best Initial V0: %.4f m/s\n', Best_V0_Vx);
fprintf('Best Launch Elevation: %.4f°\n', Best_Theta_Vx);
fprintf('Corresponding Max Horizontal Kinetic Energy (Kx): %.2f J\n',
Max_Kx);
% --- Run Max Vy Optimization (Index 4) ---
fprintf('\n=== 16. Maximize Terminal Vertical Speed (Vy) Sweep ===\n');
[Max_Vy, Best_V0_Vy, Best_Theta_Vy, V_log_Vy, V_terminal_log_Vy] =
sweep_for_max_terminal_speed(params_base, 4, V_start_opt, V_end_opt,
Step_coarse, Step_fine);
Max_Ky = 0.5 * m * Max_Vy^2;
fprintf('MAX VY RESULT:\n');
fprintf('Max Terminal Vy: %.4f m/s\n', Max_Vy);
fprintf('Best Initial V0: %.4f m/s\n', Best_V0_Vy);
fprintf('Best Launch Elevation: %.4f°\n', Best_Theta_Vy);
```

```
fprintf('Corresponding Max Vertical Kinetic Energy (Ky): %.2f J\n',
Max_Ky);
%% 17. Max Vx Trajectory and Locus Plot (Figure 6)
if ~isnan(Best_V0_Vx)
  figure('Name', 'Figure 6: Maximize Terminal Horizontal Speed (Vx)');
  Y_end_Vx = get_terminal_state(Best_V0_Vx, deg2rad(Best_Theta_Vx),
params_base);
  Y0_Vx = [0; Y_initial; Best_V0_Vx * cos(deg2rad(Best_Theta_Vx));
Best_V0_Vx * sin(deg2rad(Best_Theta_Vx))];
  [T_Vx, Y_Vx] = ode113(@(t, Y) shell_dynamics(t, Y, params_base), [0,
200], Y0_Vx, ode_options);
  Vx_traj = Y_Vx(:, 3);
    % Subplot 1: Trajectory
  subplot(2, 1, 1);
  hold on;
  plot(Y_Vx(:, 1), Y_Vx(:, 2), 'b-', 'LineWidth', 2, 'DisplayName',
sprintf('Max $V_x$ Trajectory ($%.2f^\\circ$)', Best_Theta_Vx));
  plot(X_target, Y_initial, 'rp', 'MarkerSize', 10, 'DisplayName', 'Target
Point');
  plot(Y_end_Vx(1), Y_end_Vx(2), 'bo', 'MarkerSize', 4, 'DisplayName',
sprintf('Impact ($V_x$=%.2f m/s)', Y_end_Vx(3)));
  title(sprintf('Max Terminal Horizontal Speed $V_x$, Trajectory:
$V_0$=%.2f m/s', Best_V0_Vx), 'Interpreter', 'latex');
  xlabel('Horizontal Distance $X$ (m)', 'Interpreter', 'latex');
  ylabel('Altitude $Y$ (m)', 'Interpreter', 'latex');
  grid on;
  axis tight;
  legend('show', 'Location', 'NorthEast', 'Interpreter', 'latex');
    % Subplot 2: Vx vs V_0
  subplot(2, 1, 2);
  plot(V_log_Vx, V_terminal_log_Vx, 'b.', 'MarkerSize', 8, 'DisplayName',
'$V_x$ at Target');
  hold on;
  scatter(Best_V0_Vx, Max_Vx, 50, 'r', 'filled', 'DisplayName',
sprintf('Max $V_x$: %.2f m/s', Max_Vx));
  title('Terminal Horizontal Speed $V_x$ vs. Initial Speed $V_0$',
'Interpreter', 'latex');
  xlabel('Initial Speed $V_0$ (m/s)', 'Interpreter', 'latex');
  ylabel('Terminal Horizontal Speed $V_x$ (m/s)', 'Interpreter', 'latex');
  grid on;
  axis tight;
  legend('show', 'Location', 'NorthWest', 'Interpreter', 'latex');
end
%% 18. Max Vy Trajectory and Locus Plot (Figure 7)
```

```
if ~isnan(Best_V0_Vy)
  figure('Name', 'Figure 7: Maximize Terminal Vertical Speed (Vy)');
  Y_end_Vy = get_terminal_state(Best_V0_Vy, deg2rad(Best_Theta_Vy),
params_base);
  Y0_Vy = [0; Y_initial; Best_V0_Vy * cos(deg2rad(Best_Theta_Vy));
Best_V0_Vy * sin(deg2rad(Best_Theta_Vy))];
  [T_Vy, Y_Vy] = ode113(@(t, Y) shell_dynamics(t, Y, params_base), [0,
200], Y0_Vy, ode_options);
  Vy_traj = Y_Vy(:, 4);
    % Subplot 1: Trajectory
  subplot(2, 1, 1);
  hold on;
  plot(Y_Vy(:, 1), Y_Vy(:, 2), 'r-', 'LineWidth', 2, 'DisplayName',
sprintf('Max $V_y$ Trajectory ($%.2f^\\circ$)', Best_Theta_Vy));
  plot(X_target, Y_initial, 'rp', 'MarkerSize', 10, 'DisplayName', 'Target
Point');
  plot(Y_end_Vy(1), Y_end_Vy(2), 'ro', 'MarkerSize', 4, 'DisplayName',
sprintf('Impact ($|V_y|$=%.2f m/s)', abs(Y_end_Vy(4))));
  title(sprintf('Max Terminal Vertical Speed $|V_y|$ Trajectory: $V_0$=%.2f
m/s', Best_V0_Vy), 'Interpreter', 'latex');
  xlabel('Horizontal Distance $X$ (m)', 'Interpreter', 'latex');
  ylabel('Height $Y$ (m)', 'Interpreter', 'latex');
  grid on;
  axis tight;
  legend('show', 'Location', 'NorthEast', 'Interpreter', 'latex');
    % Subplot 2: |Vy| vs V_0
  subplot(2, 1, 2);
  plot(V_log_Vy, V_terminal_log_Vy, 'r.', 'MarkerSize', 8, 'DisplayName',
'$|V_y|$ at Target');
  hold on;
  scatter(Best_V0_Vy, Max_Vy, 50, 'r', 'filled', 'DisplayName',
sprintf('Max $|V_y|$: %.2f m/s', Max_Vy));
  title('Terminal Vertical Speed $|V_y|$ vs. Initial Velocity $(V_0)$',
'Interpreter', 'latex');
  xlabel('Initial Speed $V_0$ (m/s)', 'Interpreter', 'latex');
  ylabel('Terminal Vertical Speed $|V_y|$ (m/s)', 'Interpreter', 'latex');
  grid on;
  legend('show', 'Location', 'NorthWest', 'Interpreter', 'latex');
  axis tight;
end
%% 19. Combined Critical Trajectories (Figure 8)
figure('Name', 'Figure 8: Max V_x, Max |V_y|, and Min V_0 Trajectories');
hold on;
```

```
plot(X_target, Y_initial, 'rp', 'MarkerSize', 15, 'DisplayName', 'Target
Point (1200m, 500m)');
% V_0_min
if ~isempty(Y_crit_low)
  plot(Y_crit_low(:, 1), Y_crit_low(:, 2), 'k:', 'LineWidth', 2,
'DisplayName', sprintf('Min $V_0$ Trajectory ($V_0$=%.2f)',
v_0_min_found));
end
% Max Vx
if ~isempty(Y_Vx)
  plot(Y_Vx(:, 1), Y_Vx(:, 2), 'b-', 'LineWidth', 2, 'DisplayName',
sprintf('Max $V_x$ Trajectory ($V_0$=%.2f)', Best_V0_Vx));
end
% Max Vy
if ~isempty(Y_Vy)
  plot(Y_Vy(:, 1), Y_Vy(:, 2), 'r--', 'LineWidth', 2, 'DisplayName',
sprintf('Max $|V_y|$ Trajectory ($V_0$=%.2f)', Best_V0_Vy));
end
title('Comparison of Critical Trajectories (Min $V_0$, Max $V_x$, Max
$|V_y|$)', 'Interpreter', 'latex');
xlabel('Horizontal Distance $X$ (m)', 'Interpreter', 'latex');
ylabel('Height $Y$ (m)', 'Interpreter', 'latex');
axis tight;
grid on;
legend('show', 'Location', 'NorthEast', 'Interpreter', 'latex');
    hold off;
```

10.5 Appendix E: Ballistic_Simulation.m[2]

```
%% Artillery Correction Factor and Range Table Generator
%% 1. Physical Constants and Initial Conditions Definition
% Projectile Parameters
d = 0.11;                   % Diameter (m)
m = 5;                      % Mass (kg)
A = pi * (d/2)^2;           % Cross-sectional Area (m^2)
C_D_CRIT_M0 = 0.42;         % Drag crisis minimum at Mach 0.
% Environmental Parameters
g = 9.81;                   % Gravitational Acceleration (m/s^2) - Assumed
constant
H_ref = 500;                % Reference Altitude (m)
rho_ref = 1.20;             % Standard Air Density (kg/m^3) at H_ref=500m
(for context)
```

[2]Generative AI Assisted Coding on program framework, notes, variable naming, integration of separate code documents, debugging. Algorithms including iterative solution method and modelling are not included. Using Google Gemini 2.5-Flash.

```
c0_ref = 340;               % Standard Speed of Sound (m/s) (for context)
mu_ref = 1.8e-5;            % Air Dynamic Viscosity (Pa*s) (for context)
% Calculation Parameters for Sweeps and Corrections
X_std = 1200;               % Standard Range (m) for Cx, Ch, Cw, Cz
calculation
X_buffer = 10;              % Range buffer for linear Cx approximation (+/-
10m)
H_std = 500;                % Standard Altitude Reference (m)
H_delta = 500;              % Altitude change for linear Ch approximation
(H_std + 500m)
W_std = 10;
% Initial conditions for sweep
V0_list = [300, 350, 400, 450]; % Initial Velocities (m/s)
Theta_list_deg = 1:1:89;       % Launch Angles (degrees)
% High-Precision Solver Setups
global ode_options fzero_options
% RelTol 1e-9, AbsTol 1e-11 for high accuracy integration.
ode_options = odeset('RelTol', 1e-9, 'AbsTol', 1e-11, 'Events',
@event_function);
% Tighten root finding tolerance to 1e-12 radians for launch angles.
fzero_options = optimset('TolX', 1e-12, 'Display', 'off');

%% 2. Dynamic ISA Atmosphere Model
function [rho, c0, mu] = isa_atmosphere(H)
    % Sea-level constants (h=0m)
    T0 = 288.15;        % Temperature (K)
    P0 = 101325;        % Pressure (Pa)
    rho0 = 1.225;       % Density (kg/m^3)
    mu_ref = 1.7894e-5; % Reference Viscosity (Pa*s) at T_ref
    T_ref = 288.15;     % Reference Temperature for viscosity (K)
    L = -0.0065;        % Temperature Lapse Rate (K/m) for troposphere
    gamma = 1.4;        % Ratio of specific heats
    R = 287.05;         % Specific Gas Constant (J/(kg*K))
    g0 = 9.80665;       % Gravity (m/s^2)
    H_trop = 11000;     % Tropopause altitude (m)
    S_mu = 110.4;       % Sutherland's Constant (K)

    H_actual = H; % Use instantaneous altitude

    if H_actual < 0
        H_actual = 0; % Prevent errors below ground
    end

    if H_actual <= H_trop % Troposphere (up to 11 km)
```

```
        % 1. Temperature
        T = T0 + L * H_actual;

        % 2. Pressure (using P/P0 = (T/T0)^(-g0 / (L*R)))
        P = P0 * (T / T0)^(-g0 / (L * R));

    else % Stratosphere (Isothermal layer above 11km)
        T_trop = T0 + L * H_trop;
        P_trop = P0 * (T_trop / T0)^(-g0 / (L * R));

        T = T_trop; % Temperature is constant
        % Pressure (using P/P_trop = exp(-g0 * (H-H_trop) / (R*T_trop)))
        P = P_trop * exp(-g0 * (H_actual - H_trop) / (R * T_trop));
    end

    % 3. Calculate Speed of Sound (c0 = sqrt(gamma * R * T))
    c0 = sqrt(gamma * R * T);

    % 4. Calculate Dynamic Viscosity (Sutherland's Law)
    mu = mu_ref * (T / T_ref)^1.5 * ((T_ref + S_mu) / (T + S_mu));

    % 5. Calculate Density (rho = P / (R * T))
    rho = P / (R * T);

    rho = max(rho, 0.0); % Ensure non-negative density
end

%% 3. Drag Coefficient Cx(M, Re) Model (Shared by 2D and 3D Dynamics)
function Cx = improved_Cx(v, c0, rho, d, mu, C_D_CRIT_M0)

    if v < 1e-9
        Cx = C_D_CRIT_M0;
        return;
    end

    M = v / c0;
    Re = (rho * v * d) / mu;

    % H_M calculation (Equations 8d and 8e) - Used for C_M
    if M < 1
        H_M = 0.0239*M^3 + 0.212*M^2 - 0.074*M + 1;
    else % M >= 1
        H_M = 0.93 + 1 / (3.5 + M^5);
    end
```

```
    C_M = H_M;

    % G_M calculation (Equations 8b and 8c)
    if M < 0.8
        G_M = 166*M^3 + 3.29*M^2 - 10.9*M + 20;
    else % M >= 0.8
        G_M = 5 + 40*(M^(-3));
    end

    % C_D calculation (Equation 8a)
    Term1 = (24/Re) * (1 + 0.15*Re^0.687) * H_M;
    Denominator = 1 + (42500 / (Re^(1.16*C_M))) + (G_M / (Re^0.5));
    Term2 = (0.42 * C_M) / Denominator;

    Cx = Term1 + Term2;
    Cx = max(Cx, 0.01);
end

%% 4. Point Mass Dynamics Equation (2D: Dynamic Atmosphere)
% State Vector Y = [x; y; vx; vy]
function dYdt = shell_dynamics(~, Y, params)
    y = Y(2); % Altitude
    vx = Y(3);
    vy = Y(4);


    H_actual = y + params.Atmosphere_Offset_m;
    [rho, c0, mu] = isa_atmosphere(H_actual);

    m = params.m;
    A = params.A;
    g = params.g;
    d = params.d;
    C_D_CRIT_M0 = params.C_D_CRIT_M0;

    v = sqrt(vx^2 + vy^2);
    if v < 1e-9
        dYdt = [vx; vy; 0; -g];
        return;
    end

    % Cx calculation uses the DYNAMIC c0, rho, mu
    Cx = improved_Cx(v, c0, rho, d, mu, C_D_CRIT_M0);
    drag_accel_factor = -(0.5 * rho * A * Cx * v) / m;
```

```
    % Acceleration components
    ax = drag_accel_factor * vx;
    ay = -g + drag_accel_factor * vy;
    dYdt = [vx; vy; ax; ay];
end

%% 5. Point Mass Dynamics Equation (3D: Dynamic Atmosphere)
% State Vector Y = [x; y; z; vx; vy; vz]
function dYdt = shell_dynamics_3d(~, Y, params)
    y = Y(2); % Altitude
    vx = Y(4);
    vy = Y(5);
    vz = Y(6);
    H_actual = y + params.Atmosphere_Offset_m;
    [rho, c0, mu] = isa_atmosphere(H_actual);
    vx_rel = vx - params.Wind_Vector(1);
    vy_rel = vy - params.Wind_Vector(2);
    vz_rel = vz - params.Wind_Vector(3);
    v_rel = sqrt(vx_rel^2 + vy_rel^2 + vz_rel^2);
    Cx = improved_Cx(v_rel, c0, rho, params.d, mu, params.C_D_CRIT_M0);

    % Drag acceleration magnitude factor
    if v_rel < 1e-9
        drag_accel_factor = 0;
    else
        drag_accel_factor = -(0.5 * rho * params.A * Cx * v_rel) / params.m;
    end
    ax = drag_accel_factor * vx_rel;
    ay = -params.g + drag_accel_factor * vy_rel;
    az = drag_accel_factor * vz_rel;

    dYdt = [vx; vy; vz; ax; ay; az];
end

%% 6. Event Handler Functions
% 2D Event Handler (Stops when Y=0, i.e., impact - for shell_dynamics)
function [value, isterminal, direction] = event_function(~, Y, ~)
    value = Y(2);      % Stop condition: Y=0 (Altitude)
    isterminal = 1;    % Stop integration
    direction = -1;    % Falling only
end
% 3D Event Handler (Stops when Y=0, i.e., impact - for shell_dynamics_3d)
function [value, isterminal, direction] = event_function_3d(~, Y, ~)
```

```
    value = Y(2);      % Stop condition: Y=0 (Altitude)
    isterminal = 1;    % Stop integration
    direction = -1;    % Falling only
end

%% 7. Angle Finding Helper Functions
% Error function for fzero (finds the angle that hits X_target) - 2D
function error = find_angle_error(theta, params)
    global ode_options
    ode_options.Events = @event_function; % Ensure correct 2D event handler
    v0 = params.v0;
    Y0 = [0; 0; v0 * cos(theta); v0 * sin(theta)];

    [~, Y] = ode113(@(t, Y) shell_dynamics(t, Y, params), [0, 200], Y0,
ode_options);

    if isempty(Y) || Y(end, 2) > 0.1
        Range = NaN; % Failed to hit ground properly
    else
        Range = Y(end, 1);
    end

    if isnan(Range)
        error = inf;
    else
        error = Range - params.X_target;
    end
end
% Helper function to find two launch angles for a given V0 - 2D
function [theta_low_rad, theta_high_rad] = find_launch_angles(v0, X_target,
params_base)
    global fzero_options
    params = params_base;
    params.v0 = v0;
    params.X_target = X_target;

    theta_low_rad = NaN;
    theta_high_rad = NaN;

    % Search for Low Angle (Flat Trajectory)
    try
        theta_low_rad = fzero(@(theta) find_angle_error(theta, params),
[deg2rad(0.01), deg2rad(45)], fzero_options);
        if abs(find_angle_error(theta_low_rad, params)) > 1e-3
```

```
            theta_low_rad = NaN;
        end
    catch; end

    % Search for High Angle (Lobe Trajectory)
    try
        theta_high_rad = fzero(@(theta) find_angle_error(theta, params),
[deg2rad(45), deg2rad(89.99)], fzero_options);
        if abs(find_angle_error(theta_high_rad, params)) > 1e-3
            theta_high_rad = NaN;
        end
    catch; end
end
% Error function for fzero (finds the angle that hits X_target) - 3D
function error = find_angle_error_3d(theta, params)
    global ode_options
    ode_options.Events = @event_function_3d; % Ensure correct 3D event
handler
    v0 = params.v0;
    Y0 = [0; 0; 0; v0 * cos(theta); v0 * sin(theta); 0];

    [~, Y] = ode113(@(t, Y) shell_dynamics_3d(t, Y, params), [0, 200], Y0,
ode_options);

    Range = NaN;
    if ~isempty(Y)
        Range = Y(end, 1); % X-coordinate at impact
    end
    if isnan(Range)
        error = inf;
    else
        error = Range - params.X_target;
    end
end
% Helper function to find two launch angles for a given V0 - 3D
function [theta_low_rad, theta_high_rad] = find_launch_angles_3d(v0,
X_target, params_base)
    global fzero_options
    params = params_base;
    params.v0 = v0;
    params.X_target = X_target;

    theta_low_rad = NaN;
    theta_high_rad = NaN;
```

```
    % Search for Low Angle (Flat Trajectory)
    try
        theta_low_rad = fzero(@(theta) find_angle_error_3d(theta, params),
[deg2rad(0.01), deg2rad(45)], fzero_options);
        if abs(find_angle_error_3d(theta_low_rad, params)) > 1e-3
            theta_low_rad = NaN;
        end
    catch; end

    % Search for High Angle (Lobe Trajectory)
    try
        theta_high_rad = fzero(@(theta) find_angle_error_3d(theta, params),
[deg2rad(45), deg2rad(89.99)], fzero_options);
        if abs(find_angle_error_3d(theta_high_rad, params)) > 1e-3
            theta_high_rad = NaN;
        end
    catch; end
end


%% TASK 1: Artillery Range Table and Cx Factor Generator
% Package parameters for use in 2D solver functions (Dynamic Model Base)
params_base_2d.m = m;
params_base_2d.A = A;
params_base_2d.g = g;
params_base_2d.d = d;
params_base_2d.C_D_CRIT_M0 = C_D_CRIT_M0;
params_base_2d.Atmosphere_Offset_m = 0; % Standard ISA profile (offset 0)
% Data structure for output
Range_Matrix = zeros(length(Theta_list_deg), length(V0_list));

fprintf('\n================================================================
==\n');
fprintf('       Task 1: Calculating Artillery Range Table and Cx\n');
fprintf('       (Dynamic Atmosphere Model Used)\n');
fprintf('================================================================
\n');
num_v0 = length(V0_list);
num_theta = length(Theta_list_deg);

% Ensure 2D event handler is set for the sweep
ode_options.Events = @event_function;
```

```
% --- 7.1. Range Table Sweep ---
for i = 1:num_theta
    theta_deg = Theta_list_deg(i);
    theta_rad = deg2rad(theta_deg);

    if mod(theta_deg, 10) == 1
        fprintf('Processing angle %d of %d...\n', theta_deg,
max(Theta_list_deg));
    end

    for j = 1:num_v0
        v0 = V0_list(j);

        Y0 = [0; 0; v0 * cos(theta_rad); v0 * sin(theta_rad)];

        [~, Y] = ode113(@(t, Y) shell_dynamics(t, Y, params_base_2d), [0,
200], Y0, ode_options);

        Range = NaN;
        if ~isempty(Y)
            Range = Y(end, 1);
        end

        Range_Matrix(i, j) = Range;
    end
end
fprintf('Range Table calculation complete. Total %d trajectories
solved.\n', num_v0 * num_theta);

% --- 7.2. Range Table Output ---
Angle_Column = Theta_list_deg';
V0_Headers = V0_list;
Headers = ['Angle (deg)', arrayfun(@(v) sprintf('V0=%d (m/s)', v),
V0_Headers, 'UniformOutput', false)];
Output_Table_Data = [Angle_Column, Range_Matrix];
Output_Table = array2table(Output_Table_Data, 'VariableNames', Headers);
Excel_Filename_Range = 'Range_Table.xlsx';
writetable(Output_Table, Excel_Filename_Range, 'WriteMode', 'replacefile');
fprintf('\nSUCCESS: Range Table saved to "%s"\n', Excel_Filename_Range);

% --- 7.3. Range Correction Factor (Cx) Calculation ---
% (Logic unchanged, still uses Range_Matrix)
X_interp_low = X_std - X_buffer;
X_interp_high = X_std + X_buffer;
```

```
Delta_X = X_interp_high - X_interp_low; % 20m

Cx_Factors = table('Size', [length(V0_list), 3], ...
    'VariableTypes', {'double', 'double', 'double'}, ...
    'VariableNames', {'V0_ms', 'Cx_low_deg_per_m', 'Cx_high_deg_per_m'});

fprintf('\nCalculating Range Correction Factors (Cx) at X_std = %.0f m\n',
X_std);
for j = 1:length(V0_list)
    v0 = V0_list(j);
    Cx_Factors.V0_ms(j) = v0;
    Ranges = Range_Matrix(:, j);
    Angles = Theta_list_deg';

    % Low Angle Correction
    [~, idx_max_range] = max(Ranges);
    idx_max_range = min(idx_max_range, find(Angles >= 45, 1));
    Ranges_low = Ranges(1:idx_max_range);
    Angles_low = Angles(1:idx_max_range);

    if max(Ranges_low) >= X_interp_high && min(Ranges_low) <= X_interp_low
        try
            theta_low_1190 = interp1(Ranges_low, Angles_low, X_interp_low,
'linear', 'extrap');
            theta_low_1210 = interp1(Ranges_low, Angles_low, X_interp_high,
'linear', 'extrap');
            Cx_low = (theta_low_1210 - theta_low_1190) / Delta_X;
            Cx_Factors.Cx_low_deg_per_m(j) = Cx_low;
        catch ME
            fprintf('Warning for V0=%d (Low Angle): %s\n', v0, ME.message);
            Cx_Factors.Cx_low_deg_per_m(j) = NaN;
        end
    else
        % Suppress warning if range clearly cannot be reached
        if max(Ranges_low) < X_interp_low
            fprintf('V0=%d (Low Angle) max range (%.0fm) too short for Cx
calculation at %.0fm.\n', v0, max(Ranges_low), X_std);
        else
            fprintf('V0=%d (Low Angle) range interpolation failed
around %.0fm. Cx is NaN.\n', v0, X_std);
        end
        Cx_Factors.Cx_low_deg_per_m(j) = NaN;
    end
```

```
    % High Angle Correction
    Ranges_high = Ranges(idx_max_range:end);
    Angles_high = Angles(idx_max_range:end);
    [Ranges_high_sorted, sort_idx] = sort(Ranges_high, 'ascend');
    Angles_high_sorted = Angles_high(sort_idx);

    if max(Ranges_high_sorted) >= X_interp_high && min(Ranges_high_sorted)
<= X_interp_low
        try
            theta_high_1190 = interp1(Ranges_high_sorted,
Angles_high_sorted, X_interp_low, 'linear', 'extrap');
            theta_high_1210 = interp1(Ranges_high_sorted,
Angles_high_sorted, X_interp_high, 'linear', 'extrap');
            Cx_high = (theta_high_1210 - theta_high_1190) / Delta_X;
            Cx_Factors.Cx_high_deg_per_m(j) = Cx_high;
        catch ME
             fprintf('Warning for V0=%d (High Angle): %s\n', v0,
ME.message);
            Cx_Factors.Cx_high_deg_per_m(j) = NaN;
        end
    else
        % Suppress warning if range clearly cannot be reached
         if max(Ranges_high_sorted) < X_interp_low
            % Do nothing, V0 too short
         else
            fprintf('V0=%d (High Angle) range interpolation failed
around %.0fm. Cx is NaN.\n', v0, X_std);
         end
        Cx_Factors.Cx_high_deg_per_m(j) = NaN;
    end
end

Excel_Corr_Filename_Cx = 'Range_Correction_Factors.xlsx';
writetable(Cx_Factors, Excel_Corr_Filename_Cx, 'WriteMode', 'replacefile');
fprintf('\n--- Range Correction Factors (Cx) ---\n');
disp(Cx_Factors);
fprintf('\nSUCCESS: Range Correction Factors saved to "%s"\n',
Excel_Corr_Filename_Cx);
fprintf('================================================================
\n');

%% TASK 2: Altitude Correction Factor (Ch) Generator
% Standard and alternative altitude offsets for dynamic ISA model
```

```
Atm_Offset_std = H_std; % Offset = 500m (Simulates H=500m conditions at
Y=0)
Atm_Offset_alt = H_std + H_delta; % Offset = 1000m (Simulates H=1000m
conditions at Y=0)

fprintf('\n================================================================
==\n');
fprintf('  Task 2: Calculating Altitude Correction Factors (Ch)\n');
fprintf('  (Dynamic ISA with Profile Offsets)\n');
fprintf('================================================================
\n');
fprintf('  Standard Atmosphere Offset: %.0f m\n', Atm_Offset_std);
fprintf('  Alternative Atmosphere Offset: %.0f m\n', Atm_Offset_alt);
fprintf('  Target Range X_std: %.0f m\n', X_std);

% Initialize storage for angles
Theta_std_low = zeros(size(V0_list));
Theta_std_high = zeros(size(V0_list));
Theta_alt_low = zeros(size(V0_list));
Theta_alt_high = zeros(size(V0_list));

% Package static parameters template
params_template_ch.m = m;
params_template_ch.A = A;
params_template_ch.g = g;
params_template_ch.d = d;
params_template_ch.C_D_CRIT_M0 = C_D_CRIT_M0;

% --- 7.1. Angle Solving at Standard and Alternate Offsets (2D) ---
for j = 1:length(V0_list)
    v0 = V0_list(j);

    % Standard Case (Atmosphere Offset = H_std)
    params_std_ch = params_template_ch;
    params_std_ch.Atmosphere_Offset_m = Atm_Offset_std;
    [t_l_std, t_h_std] = find_launch_angles(v0, X_std, params_std_ch);

    Theta_std_low(j) = rad2deg(t_l_std);
    Theta_std_high(j) = rad2deg(t_h_std);

    % Alternative Case (Atmosphere Offset = H_std + H_delta)
    params_alt_ch = params_template_ch;
    params_alt_ch.Atmosphere_Offset_m = Atm_Offset_alt;
    [t_l_alt, t_h_alt] = find_launch_angles(v0, X_std, params_alt_ch);
```

```
    Theta_alt_low(j) = rad2deg(t_l_alt);
    Theta_alt_high(j) = rad2deg(t_h_alt);

    fprintf('V0=%.0f m/s: \n', v0);
    fprintf('  Std Offset (%.0fm): Low Angle=%.4f°, High Angle=%.4f°\n',
Atm_Offset_std, Theta_std_low(j), Theta_std_high(j));
    fprintf('  Alt Offset (%.0fm): Low Angle=%.4f°, High Angle=%.4f°\n',
Atm_Offset_alt, Theta_alt_low(j), Theta_alt_high(j));
end

% --- 7.2. Altitude Correction Factor (Ch) Calculation ---
Delta_H_m = H_delta; % H_alt - H_std = 500m
Factor_HM = 100; % Conversion factor for "per 100 meters"

Ch_Factors = table('Size', [length(V0_list), 3], ...
    'VariableTypes', {'double', 'double', 'double'}, ...
    'VariableNames', {'V0_ms', 'Ch_low_deg_per_HM', 'Ch_high_deg_per_HM'});

for j = 1:length(V0_list)
    v0 = V0_list(j);
    Ch_Factors.V0_ms(j) = v0;

    % Low Angle Correction (C_H,low)
    delta_theta_low = Theta_alt_low(j) - Theta_std_low(j);
    if isnan(delta_theta_low) || isnan(Theta_std_low(j))
        Ch_low = NaN;
    else
        Ch_low = (delta_theta_low / Delta_H_m) * Factor_HM;
    end
    Ch_Factors.Ch_low_deg_per_HM(j) = Ch_low;

    % High Angle Correction (C_H,high)
    delta_theta_high = Theta_alt_high(j) - Theta_std_high(j);
    if isnan(delta_theta_high) || isnan(Theta_std_high(j))
        Ch_high = NaN;
    else
        Ch_high = (delta_theta_high / Delta_H_m) * Factor_HM;
    end
    Ch_Factors.Ch_high_deg_per_HM(j) = Ch_high;

    fprintf('V0=%.0f m/s: C_H,low = %.6f deg/HM | C_H,high = %.6f
deg/HM\n', v0, Ch_low, Ch_high);
    if isnan(Ch_low)
```

```
        fprintf('  WARNING: V0=%.0f m/s cannot reach %.0f m range in one of
the cases, C_H is invalid.\n', v0, X_std);
    end
end

% --- 7.3. Output to Excel ---
Excel_Corr_Filename_Ch = 'Altitude_Correction_Factors.xlsx';
writetable(Ch_Factors, Excel_Corr_Filename_Ch, 'WriteMode', 'replacefile');
fprintf('\n--- Altitude Correction Factors (C_H) ---\n');
disp(Ch_Factors);
fprintf('\nSUCCESS: Altitude Correction Factors saved to "%s"\n',
Excel_Corr_Filename_Ch);
fprintf('================================================================
\n');

%% TASK 3: Wind Correction Factor (Cw, Cz) Generator
fprintf('\n================================================================
==\n');
fprintf('  Task 3: Calculating Wind Correction Factors (Cw, Cz)\n');
fprintf('  (Dynamic Atmosphere Model Used)\n');
fprintf('================================================================
\n');
fprintf('  Target Range X_std: %.0f m\n', X_std);
fprintf('  Standard Wind Magnitude W_std: %.0f m/s\n', W_std);

% Package parameters for 3D solver functions (Dynamic Model Base)
params_base_3d.m = m;
params_base_3d.A = A;
params_base_3d.g = g;
params_base_3d.d = d;
params_base_3d.C_D_CRIT_M0 = C_D_CRIT_M0;
params_base_3d.Atmosphere_Offset_m

% Initialize table for correction factors
Cw_Cz_Factors = table('Size', [length(V0_list), 5], ...
    'VariableTypes', {'double', 'double', 'double', 'double',
'double'}, ...
    'VariableNames', {'V0_ms', 'Theta_std_low_deg', 'Cz_m_per_mps',
'Cw_low_deg_per_mps', 'Cw_high_deg_per_mps'});

% --- 8.1. STAGE 1: Calculating Standard Angles (Zero Wind) ---
fprintf('\n  STAGE 1: Calculating Standard Angles (Zero Wind)\n');
params_std_wind = params_base_3d;
params_std_wind.Wind_Vector = [0, 0, 0]; % [Wx, Wy, Wz]
```

```
% Ensure 3D event handler is set for this task
ode_options.Events = @event_function_3d;

for j = 1:length(V0_list)
    v0 = V0_list(j);
    Cw_Cz_Factors.V0_ms(j) = v0;

    % Only the low angle is required for the output table, but need both
for Cw high later
    [t_l_std, t_h_std] = find_launch_angles_3d(v0, X_std, params_std_wind);
    theta_std_low_deg = rad2deg(t_l_std);

    Cw_Cz_Factors.Theta_std_low_deg(j) = theta_std_low_deg;
    fprintf('V0=%.0f m/s: Standard Low Angle (%.0fm target): %.4f
degrees\n', v0, X_std, theta_std_low_deg);
end

% --- 8.2. STAGE 2: Side Wind Deviation (C_Z) ---
fprintf('\n  STAGE 2: Calculating Side Wind Deviation (C_Z)\n');
params_side_wind = params_base_3d;
params_side_wind.Wind_Vector = [0, 0, W_std]; % Side wind along Z-axis

for j = 1:length(V0_list)
    v0 = V0_list(j);
    theta_std_low_rad = deg2rad(Cw_Cz_Factors.Theta_std_low_deg(j));

    if isnan(theta_std_low_rad)
        Cw_Cz_Factors.Cz_m_per_mps(j) = NaN;
        fprintf('V0=%.0f m/s: C_Z calculation skipped (Cannot reach target
range).\n', v0);
        continue;
    end

    Y0 = [0; 0; 0; v0 * cos(theta_std_low_rad); v0 *
sin(theta_std_low_rad); 0];
    [~, Y] = ode113(@(t, Y) shell_dynamics_3d(t, Y, params_side_wind), [0,
200], Y0, ode_options);

    Z_impact = NaN;
    if ~isempty(Y)
        Z_impact = Y(end, 3); % Z-coordinate (lateral deviation) at impact
    end
```

```
    % C_Z = Z_impact / W_std (meters per meter/second)
    Cz = Z_impact / W_std;
    Cw_Cz_Factors.Cz_m_per_mps(j) = Cz;

    fprintf('V0=%.0f m/s: Z_impact=%.4f m | C_Z = %.6f m/(m/s)\n', v0,
Z_impact, Cz);
end

% --- 8.3. STAGE 3: Head/Tail Wind Correction (C_W) ---
fprintf('\n  STAGE 3: Calculating Longitudinal Wind Correction (C_W)\n');
W_longitudinal = -W_std; % Head wind (negative X direction)
params_head_wind = params_base_3d;
params_head_wind.Wind_Vector = [W_longitudinal, 0, 0];

for j = 1:length(V0_list)
    v0 = V0_list(j);
    theta_std_low_deg = Cw_Cz_Factors.Theta_std_low_deg(j);

    if isnan(theta_std_low_deg)
        Cw_Cz_Factors.Cw_low_deg_per_mps(j) = NaN;
        Cw_Cz_Factors.Cw_high_deg_per_mps(j) = NaN;
        fprintf('V0=%.0f m/s: C_W calculation skipped (Cannot reach target
range).\n', v0);
        continue;
    end

    % Find angles for head-wind condition
    [t_l_head, t_h_head] = find_launch_angles_3d(v0, X_std,
params_head_wind);

    theta_head_low_deg = rad2deg(t_l_head);
    theta_head_high_deg = rad2deg(t_h_head);

    % C_W,low Calculation
    delta_theta_low = theta_head_low_deg - theta_std_low_deg;
    Cw_low = delta_theta_low / W_longitudinal;
    Cw_Cz_Factors.Cw_low_deg_per_mps(j) = Cw_low;


    [~, t_h_std] = find_launch_angles_3d(v0, X_std, params_std_wind);
    theta_std_high_deg = rad2deg(t_h_std);

    delta_theta_high = theta_head_high_deg - theta_std_high_deg;
    Cw_high = delta_theta_high / W_longitudinal;
```

```
    Cw_Cz_Factors.Cw_high_deg_per_mps(j) = Cw_high;

    fprintf('V0=%.0f m/s: C_W,low = %.6f deg/(m/s) | C_W,high = %.6f
deg/(m/s)\n', v0, Cw_low, Cw_high);
end


Cw_Cz_Factors.Properties.VariableNames{'Cz_m_per_mps'} =
'Cz_lateral_m_per_mps';
Cw_Cz_Factors.Properties.VariableNames{'Cw_low_deg_per_mps'} =
'Cw_low_deg_per_mps';
Cw_Cz_Factors.Properties.VariableNames{'Cw_high_deg_per_mps'} =
'Cw_high_deg_per_mps';
Output_Factors = Cw_Cz_Factors(:, {'V0_ms', 'Theta_std_low_deg',
'Cz_lateral_m_per_mps', 'Cw_low_deg_per_mps', 'Cw_high_deg_per_mps'});
Excel_Corr_Filename_Wind = 'Wind_Correction_Factors.xlsx';
writetable(Output_Factors, Excel_Corr_Filename_Wind, 'WriteMode',
'replacefile');
fprintf('\n--- Wind Correction Factors ---\n');
disp(Output_Factors);
fprintf('\nSUCCESS: Wind Correction Factors saved to "%s"
(Theta_std_low_deg is now included!)\n', Excel_Corr_Filename_Wind);
fprintf('================================================================
\n');


%% Correction Factor Linearity Check
%% 1. Physical Constants and Base Parameters Definition
V0_base = 400;              % Base initial velocity for linearity test
(m/s)
X_target = 1200;            % Target distance (m)
H_std_ref = 500;            % Standard Altitude Reference (m) for C_H test
origin
global ode_options fzero_options
ode_options = odeset('RelTol', 1e-9, 'AbsTol', 1e-11, 'Events',
@event_function_3d);

fprintf('--- Correction Factor Linearity Check ---\n');
fprintf('Base V0: %.0f m/s | Target Range: %.0f m | Standard H Ref: %.0f
m\n', V0_base, X_target, H_std_ref);
%% 2. Base Parameters Setup
params_base.m = m;
params_base.A = A;
params_base.g = g;
```

```
params_base.d = d;
params_base.C_D_CRIT_M0 = C_D_CRIT_M0;
params_base.Atmosphere_Offset_m = 0; % Default: Use pure ISA (Y=0 -> H=0m)
params_base.Wind_Vector = [0, 0, 0]; % Start with zero wind

% --- Find the standard launch angles (V0_base, X_target, Pure ISA Env) ---
[t_l_std, t_h_std] = find_launch_angles_3d(V0_base, X_target, params_base);
theta_std_low_deg = rad2deg(t_l_std);
theta_std_high_deg = rad2deg(t_h_std);

fprintf('Standard Low Angle (Zero Wind, Pure ISA at 0m): %.4f degrees\n',
theta_std_low_deg);
%% 3. Linearity Check 1: Altitude Correction (C_H)
% Test Range: -500m to +500m offset from the baseline (Pure ISA, H=0m)
Delta_H_sweep = -500:100:500; % Altitude offsets in meters
Theta_corr_low = zeros(size(Delta_H_sweep));

fprintf('\nChecking Altitude Correction Linearity (C_H)...\n');
for i = 1:length(Delta_H_sweep)
    Atm_Offset = Delta_H_sweep(i);

    % Set parameters for angle finding (zero wind, variable offset)
    params_alt = params_base;
    params_alt.Atmosphere_Offset_m = Atm_Offset;

    % Find new low angle
    [t_l_alt, ~] = find_launch_angles_3d(V0_base, X_target, params_alt);

    % Calculate correction: Delta_theta = theta_alt - theta_std
    Theta_corr_low(i) = rad2deg(t_l_alt) - theta_std_low_deg;
end

% Perform Linear Fit
P_H = polyfit(Delta_H_sweep', Theta_corr_low', 1); % [Slope, Intercept]
y_fit_H = polyval(P_H, Delta_H_sweep);
R_sq_H = 1 - sum((Theta_corr_low - y_fit_H).^2) / sum((Theta_corr_low -
mean(Theta_corr_low)).^2);
C_H_fit = P_H(1) * 100; % Convert to deg/HM (degrees per 100 meters)

% Plotting C_H Linearity
figure(1);
plot(Delta_H_sweep, Theta_corr_low, 'bo', 'MarkerFaceColor', 'b',
'DisplayName', 'Computed $\Delta\theta_H$'); hold on;
```

```
plot(Delta_H_sweep, y_fit_H, 'r-', 'LineWidth', 2, 'DisplayName', 'Linear
Fit'); hold off;
xlabel('Atmosphere Offset $\Delta H$ (m)', 'Interpreter', 'latex');
ylabel('Angle Correction $\Delta\theta_H$ ($^{\circ}$)', 'Interpreter',
'latex');
title(sprintf('Altitude Correction Factor Linearity (V0=%.0fm/s)',
V0_base));
grid on;
legend('show', 'Interpreter', 'latex');
text(min(Delta_H_sweep), max(Theta_corr_low), ...
    sprintf('C_H = %.4f deg/HM\nR^2 = %.6f', C_H_fit, R_sq_H), ...
    'HorizontalAlignment', 'left', 'VerticalAlignment', 'top', 'EdgeColor',
'k', 'BackgroundColor', 'w', 'Interpreter', 'latex');
fprintf('C_H Linear Fit: Slope = %.6f deg/m, C_H = %.4f deg/HM, R^2
= %.6f\n', P_H(1), C_H_fit, R_sq_H);

%% 4. Linearity Check 2: Side Wind Correction (C_Z)
% Test Range: -20 m/s to +20 m/s (Lateral Wind Wz)
W_lateral_sweep = -20:5:20; % Wind speed in m/s
Z_impact_sweep = zeros(size(W_lateral_sweep));
theta_std_low_rad = deg2rad(theta_std_low_deg);

fprintf('\nChecking Side Wind Correction Linearity (C_Z)...\n');
for i = 1:length(W_lateral_sweep)
    W_lateral = W_lateral_sweep(i);

    % Side wind vector: [0, 0, W_lateral]. Maintain standard angle and
offset.
    params_side_wind = params_base;
    params_side_wind.Wind_Vector = [0, 0, W_lateral];

    % Initial state: use standard low angle, launch along X-Y plane
    Y0 = [0; 0; 0; V0_base * cos(theta_std_low_rad); V0_base *
sin(theta_std_low_rad); 0];

    [~, Y] = ode113(@(t, Y) shell_dynamics_3d(t, Y, params_side_wind), [0,
200], Y0, ode_options);

    if ~isempty(Y)
        Z_impact_sweep(i) = Y(end, 3);
    else
        Z_impact_sweep(i) = NaN;
    end
end
```

```
% Perform Linear Fit
P_Z = polyfit(W_lateral_sweep', Z_impact_sweep', 1); % [Slope, Intercept]
y_fit_Z = polyval(P_Z, W_lateral_sweep);
R_sq_Z = 1 - sum((Z_impact_sweep - y_fit_Z).^2) / sum((Z_impact_sweep -
mean(Z_impact_sweep)).^2);
C_Z_fit = P_Z(1); % m / (m/s)

% Plotting C_Z Linearity
figure(2);
plot(W_lateral_sweep, Z_impact_sweep, 'bo', 'MarkerFaceColor', 'b',
'DisplayName', 'Computed $Z_{impact}$'); hold on;
plot(W_lateral_sweep, y_fit_Z, 'r-', 'LineWidth', 2, 'DisplayName', 'Linear
Fit'); hold off;
xlabel('Lateral Wind $W_Z$ (m/s)', 'Interpreter', 'latex');
ylabel('Lateral Deviation $Z_{impact}$ (m)', 'Interpreter', 'latex');
title(sprintf('Side Wind Correction Factor Linearity (V0=%.0fm/s)',
V0_base));
grid on;
legend('show', 'Location', 'southeast', 'Interpreter', 'latex');
text(min(W_lateral_sweep), max(Z_impact_sweep), ...
    sprintf('C_Z = %.4f m/(m/s)\nR^2 = %.6f', C_Z_fit, R_sq_Z), ...
    'HorizontalAlignment', 'left', 'VerticalAlignment', 'top', 'EdgeColor',
'k', 'BackgroundColor', 'w', 'Interpreter', 'latex');
fprintf('C_Z Linear Fit: Slope = %.6f m/(m/s), R^2 = %.6f\n', P_Z(1),
R_sq_Z);

%% 5. Linearity Check 3: Longitudinal Wind Correction (C_W)
% Test Range: -20 m/s (Headwind) to +20 m/s (Tailwind)
W_longitudinal_sweep = -20:5:20; % Wind speed in m/s (Wx)
Theta_corr_low_W = zeros(size(W_longitudinal_sweep));

fprintf('\nChecking Longitudinal Wind Correction Linearity (C_W)...\n');
for i = 1:length(W_longitudinal_sweep)
    W_longitudinal = W_longitudinal_sweep(i);

    % Longitudinal wind vector: [W_longitudinal, 0, 0].
    params_long_wind = params_base;
    params_long_wind.Wind_Vector = [W_longitudinal, 0, 0];

    % Find the low angle required to hit X_target (1200m) under this wind
condition
    [t_l_wind, ~] = find_launch_angles_3d(V0_base, X_target,
params_long_wind);
```

```
    % Calculate correction: Delta_theta = theta_wind - theta_std
    Theta_corr_low_W(i) = rad2deg(t_l_wind) - theta_std_low_deg;
end

% Perform Linear Fit
P_W = polyfit(W_longitudinal_sweep', Theta_corr_low_W', 1); % [Slope,
Intercept]
y_fit_W = polyval(P_W, W_longitudinal_sweep);
R_sq_W = 1 - sum((Theta_corr_low_W - y_fit_W).^2) / sum((Theta_corr_low_W -
mean(Theta_corr_low_W)).^2);
C_W_fit = P_W(1); % deg / (m/s)

% Plotting C_W Linearity
figure(3);
plot(W_longitudinal_sweep, Theta_corr_low_W, 'bo', 'MarkerFaceColor',
'b', ...
    'DisplayName', 'Computed $\Delta\theta_W$ ($^{\circ}$)');
hold on;
plot(W_longitudinal_sweep, y_fit_W, 'r-', 'LineWidth', 2, ...
    'DisplayName', 'Linear Fit ($^{\circ}/(m/s)$)');
hold off;
xlabel('Longitudinal Wind $W_X$ (m/s)', 'Interpreter', 'latex');
ylabel('Angle Correction $\Delta\theta_W$ ($^{\circ}$)', 'Interpreter',
'latex');
title(sprintf('Longitudinal Wind Correction Linearity ($V_0=%.0f$ m/s)',
V0_base), ...
    'Interpreter', 'latex');

legend('show', 'Location', 'northwest', 'Interpreter', 'latex');
grid on;
text(max(W_longitudinal_sweep), min(Theta_corr_low_W), ...
    sprintf('$C_W = %.4f\\,^{\\circ}/(m/s)$\n$R^2=%.6f$', C_W_fit,
R_sq_W), ...
    'HorizontalAlignment', 'right', 'VerticalAlignment', 'bottom', ...
    'EdgeColor', 'k', 'BackgroundColor', 'w', 'Interpreter', 'latex');
fprintf('C_W Linear Fit: Slope = %.6f °/(m/s), R^2 = %.6f\n', P_W(1),
R_sq_W);
    fprintf('\nAll calculations complete. Check the generated Excel files
and figures for results.\n');
```