Artillery Firing Model and Manual Calculation Methods

Team 392

Problem B:Artillery

Abstract

This study tackles the ballistic challenges of 19th-century artillery by developing a method to determine the muzzle velocity and firing angle for spherical projectiles (5 kg mass, 11 cm diameter) under variable conditions. A high-fidelity ballistic model was established, incorporating air resistance which depend on altitude-varying air density and Mach number. Gravity changes is also included. For the core scenario—a target at 1200m horizontal distance with both cannon and target at 500m altitude—a bidirectional search algorithm (initial velocity range: 100–450 m/s) identified multiple valid parameter sets. Results show the minimum energy solution as 141.0 m/s at 39.6°, while an optimal balanced solution (considering economy and efficiency) is 165.0 m/s at 21.8°.

The model was extended to ranges of 1000–1500m, varying altitudes, and wind conditions. A logarithmic wind profile simulated altitude-dependent wind speed, and a hierarchical optimization algorithm minimized impact errors. This yielded a practical calculator that inputs wind speed/direction, distance, altitudes, and muzzle velocity to output elevation and azimuth angles (e.g., input: 20 m/s wind, 60° direction, 1000m distance, 500m cannon altitude, 600m target altitude, 450 m/s velocity; output: 8.02° elevation, 0.90° azimuth).

For the last practical problem, in order to enable artillery officers who have received mathematical training but do not have access to computers or calculators to quickly estimate the successful muzzle velocity and firing angle, we have designed two simple methods: the function calculation method and the table lookup method. They can choose the more accurate or simpler method according to their needs. The function for the function calculation method is as follows: elevation $= \beta_0 + \sum_{i=1}^{4} \beta_i x_i + \sum_{i=1}^{4} \sum_{j=i}^{4} \beta_{ij} x_i x_j$ azimuth $= \gamma_0 + \sum_{i=1}^{4} \gamma_i x_i$.

Keywords: Ballistics; aerodynamic drag; wind speed gradient; machine learning; rapid estimation

Contents

1 Introduction

1.1 Background

Ballistics, the physics of projectile motion, has been of paramount importance in military history. The cannons of the 19th century fired spherical solid shot, whose range and accuracy were affected by a complex interplay of factors including muzzle velocity, firing angle, air resistance, altitude, and wind. Hitting a distant target accurately required rapid estimation of appropriate firing parameters. Lacking modern computational tools, artillery officers of the era relied on experience, pre-computed firing tables, and simplified mathematical models based on classical mechanics. This problem simulates this historical context, requiring the establishment of a ballistic model for a given cannon system and the development of a rapid estimation method suitable for the technological constraints of the time.

Figure 1: Ancient artillery

1.2 Problem Restatement

Our task is to solve a targeting problem for a historical artillery piece. The specific goals are:

1.Core Calculation: For a cannon firing a spherical projectile of mass 5 kg and diameter 11 cm, with a target at a horizontal distance of 1200 meters and both the cannon and target at an altitude of 500 meters above sea level, determine the required muzzle velocity (up to 450 m/s) and firing angle above the horizontal to successfully hit the target.

2.Extended Analysis: Extend the analysis to targets at horizontal distances between 1000 and 1500 meters, at various altitudes, and account for different wind conditions.

3.Method Development: The central challenge is to develop a rapid estimation method. This method must be usable by an artillery officer trained in mathematics but lacking computers or calculators, enabling quick estimation of successful firing parameters under field conditions. The final solution must be grounded in physical principles while maintaining practicality and usability consistent with the historical context.

2 Assumptions and Notations

2.1 Assumptions

Our assumptions are as follows:

1.We employ the International Standard Atmosphere model (ISA).Assuming atmospheric conditions are at standard state, disregarding actual meteorological variations. Neglecting diurnal and seasonal variations in temperature, humidity, and air pressure, whilst assuming homogeneous atmospheric composition and disregarding pollutant effects.

2.The speed of sound is solely dependent on temperature. Using the simplified formula $v_{\text{sound}} = \sqrt{\gamma \cdot R \cdot T}$ which neglects the negligible effect of humidity on the speed of sound, is acceptable within engineering accuracy limits.

3.The drag coefficient of spherical projectiles relates solely to Mach number, disregarding rotation. The projectile is assumed smooth, rigid, and non-rotating. We take a piecewise linear function to approximate the actual drag curve. Actual drag coefficients are further influenced by surface roughness, rotation, and other factors. Drag coefficient variations across transonic regions exhibit greater complexity than the model implies, while the secondary influence of Reynolds number on drag coefficients is neglected.

4.The trajectory lies within the vertical plane. Neglecting Coriolis forces and the effects of Earth's rotation, the curvature of the Earth is negligible within the range specified in the problem. The deflection caused by Coriolis forces is approximately 0.5–1.0 metres, which also falls within the range of engineering error.

5. In the first question, the atmosphere is assumed to be stationary relative to the ground: the influence of wind speed and direction is neglected.

6.Initial Conditions: The projectile instantaneously achieves the specified muzzle velocity at the cannon's muzzle, and the firing angle is known precisely.

2.2 Notations

Table 1: Notations

Symbol	Definition	Value
d	The diameter of the shell	0.11(m)
m	The weight of the shell	5.0(kg)
g	Gravitational acceleration	9.80665(m/s^2)
h	Altitude	500(m)
A	The cross-section area of the shell	0.0095(m^2)
T_0	Standard temperature at sea level	288.15(K)
P_0	Standard pressure at sea level	101325(Pa)
ρ_0	Standard air density at sea level	1.225(kg/m^2)
L	Temperature lapse rate	0.0065(K/m)
M	Dry air molar mass	0.0289644(kg/mol)
R^*	Universal Gas Constant	8.21432(J/(mol $\cdot$ K))
R	Air gas constant	287.05J/(kg $\cdot$ K)
γ	Specific heat capacity of air	1.4

3 Model and Analysis

3.1 The solutions of ballistic calculation when the target distance is 1200 meters

3.1.1 Ideal parabolic motion and Linearized model of air resistance

Neglecting air resistance, analytical solution:

$$R = \frac{v_0^2 \sin(2\theta)}{g}, \quad T = \frac{2v_0 \sin\theta}{g}$$

Take air resistance into account,using the Siacci approximation method, the motion is decomposed into tangential and normal components:

$$\frac{dv}{dt} = -g\sin\theta - \frac{k}{m}v^2, \quad v\frac{d\theta}{dt} = -g\cos\theta$$

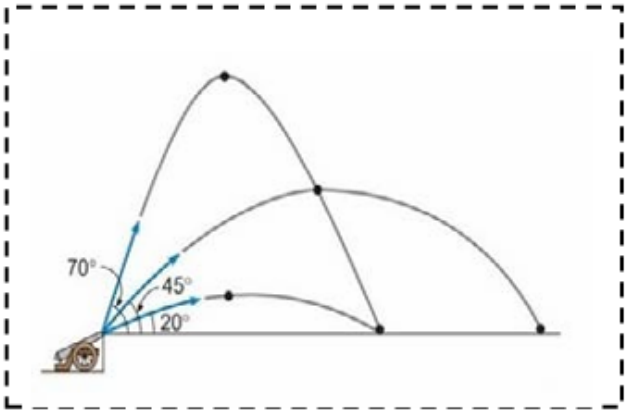

Figure 2: Motion of a Cannonball Projectile

3.1.2 Consider the impact of altitude on air resistance

· Temperature distribution: Within the troposphere (0-11,000 m), temperature decreases linearly with increasing altitude

$$T(h) = T_0 - L \cdot h$$

· Pressure distribution:

$$P(h) = P_0 \cdot \left(\frac{T(h)}{T_0}\right)^{\frac{g \cdot M}{R^* \cdot L}}$$

· Density distribution:

$$P = \rho \cdot T \cdot R^*$$

Simplified formula (for code implementation):

$$\rho = \rho_0 \times \left(1 - \frac{L \times h}{T_0}\right)^{\frac{g \cdot M}{R^* \cdot L} - 1}$$

· Sound speed calculation

The speed of sound is a function of temperature, and its calculation formula is

$$v_{\text{sound}} = \sqrt{\gamma \cdot R \cdot T}$$

· The drag coefficient C_d is a piecewise function that varies with the Mach number $M = \frac{v}{v_{\text{sound}}}$

$$C_d(M) = \begin{cases} 0.45, & M < 0.3 \\ 0.45 + 0.1 \times (M - 0.3), & 0.3 \leq M < 0.8 \\ 0.5 + 0.2 \times (M - 0.8), & 0.8 \leq M < 1.0 \\ 0.7 - 0.1 \times (M - 1.0), & 1.0 \leq M < 1.2 \\ 0.6, & M \geq 1.2 \end{cases}$$

· Air resistance formula:

$$F_{\text{drag}} = \frac{1}{2}\rho \cdot C_d \cdot Av^2$$

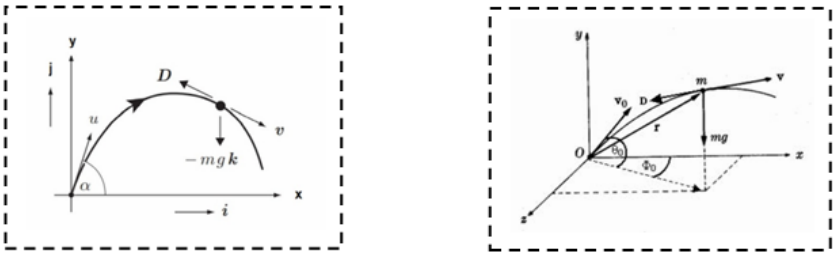

Figure 3: Force analysis of cannonballs

3.1.3 Using Python code to solve the problem

To precisely calculate the ballistic trajectory of a cannonball under the influence of air resistance and altitude, and to determine the launch parameters (initial velocity and launch angle) required to hit a target 1200 meters away, we developed a Python program. Based on a physical model, the program solves the motion equations through a search algorithm based on iterative method to find the optimal solution. The following provides a detailed description of the code implementation.[2]

· Trajectory Simulation and Impact Point Calculation

The ballistic trajectory is simulated by numerically integrating the equations of motion. The program uses Euler's method with a time step $\Delta t = 0.01\,\text{s}$ and a maximum simulation time $t_{\max} = 30\,\text{s}$. The equations of motion are:

$$\frac{dv_x}{dt} = a_x = -\frac{F_{\text{drag}}}{m} \cdot \frac{v_x}{v}, \quad \frac{dv_y}{dt} = a_y = -g - \frac{F_{\text{drag}}}{m} \cdot \frac{v_y}{v}$$

where $v = \sqrt{v_x^2 + v_y^2}$. The initial conditions are $x = 0, y = 0$, $v_x = v_0 \cos\theta$, $v_y = v_0 \sin\theta$, with θ

being the launch angle in degrees. The impact point calculation is implemented in the function calculate-range(v0, theta-deg). The program simulates the trajectory until the projectile hits the ground ($y \leq 0$), then uses linear interpolation to precisely calculate the impact x-coordinate:

$$x_{\text{impact}} = x_1 + \frac{-y_1}{y_2 - y_1}(x_2 - x_1)$$

where (x_1, y_1) and (x_2, y_2) are the last point above ground and the first point below ground, respectively. This ensures accurate range calculation.

· Search algorithm based on iterative method

To find the launch parameters that achieve the target range ($\geq 1200\,\text{m}$), the program employs two search strategies: low-to-high search and high-to-low search. This avoids local optima and improves efficiency.

Low-to-high search: Initial velocity v_0 increases from 100 m/s to 450 m/s (step size 1 m/s). For each v_0, the launch angle θ increases from 0° to 90° (step size 0.1°). Once the range exceeds 1200 m, the (v_0, θ) pair is recorded, and the inner loop breaks.

High-to-low search: Initial velocity v_0 decreases from 450 m/s to 100 m/s (step size -1 m/s). For each v_0, the launch angle θ decreases from 90° to 0° (step size -0.1°). Similarly, once the range exceeds 1200 m, the result is recorded.

Both searches call the calculate-range function to compute the range for each parameter set. After completing both searches, we arrival at a conclusion and draw the graphs of all solutions of the system of equations separately.

3.1.4 Result

The program detected that the initial velocity starts to fall within the target range when it is 141.0 m/s, and it is evident that for every initial velocity greater than 140, there are two launch angles that meet the requirements. Therefore, the results of the search from low to high angles are the smaller initial launch angles, ranges, and flight times corresponding to each speed in the range of 141-450 m/s (with a step size of 1 m/s). Correspondingly, the results of the search from high to low angles are the larger initial launch angles, ranges, and flight times corresponding to each speed in the range of 450-141 m/s. The results are shown in Figures 4 and 5.

```
Velocity 141.0 m/s, Angle 39.6°, Range 1200.1 m, Flight time 15.73 s
Velocity 142.0 m/s, Angle 36.7°, Range 1200.3 m, Flight time 14.91 s
Velocity 143.0 m/s, Angle 35.0°, Range 1200.2 m, Flight time 14.44 s
Velocity 144.0 m/s, Angle 33.7°, Range 1200.2 m, Flight time 14.09 s
Velocity 145.0 m/s, Angle 32.6°, Range 1200.1 m, Flight time 13.79 s
Velocity 146.0 m/s, Angle 31.7°, Range 1200.7 m, Flight time 13.55 s
Velocity 147.0 m/s, Angle 30.8°, Range 1200.2 m, Flight time 13.31 s
Velocity 148.0 m/s, Angle 30.1°, Range 1201.3 m, Flight time 13.13 s
Velocity 149.0 m/s, Angle 29.3°, Range 1200.2 m, Flight time 12.91 s
Velocity 150.0 m/s, Angle 28.7°, Range 1201.2 m, Flight time 12.75 s
```

Figure 4: Result from low angle to high angle

Therefore, for the first question, we draw the following conclusions:

1.We calculated the minimum initial velocity (which saves the most energy) and the corresponding muzzle angle required to hit the target. The minimum initial velocity is 141.0 m/s, and the muzzle angle is 39.6°, located at the leftmost end of the curve shown in the figure 6 below.

```
Velocity 150.0 m/s, Angle 53.7°, Range 1200.2 m, Flight time 20.39 s
Velocity 149.0 m/s, Angle 53.0°, Range 1201.0 m, Flight time 20.13 s
Velocity 148.0 m/s, Angle 52.3°, Range 1201.0 m, Flight time 19.86 s
Velocity 147.0 m/s, Angle 51.6°, Range 1200.2 m, Flight time 19.59 s
Velocity 146.0 m/s, Angle 50.7°, Range 1201.0 m, Flight time 19.27 s
Velocity 145.0 m/s, Angle 49.8°, Range 1200.6 m, Flight time 18.94 s
Velocity 144.0 m/s, Angle 48.7°, Range 1200.8 m, Flight time 18.56 s
Velocity 143.0 m/s, Angle 47.5°, Range 1200.2 m, Flight time 18.15 s
Velocity 142.0 m/s, Angle 45.8°, Range 1200.5 m, Flight time 17.60 s
Velocity 141.0 m/s, Angle 43.0°, Range 1200.0 m, Flight time 16.73 s
```

Figure 5: Result from high angle to low angle

2. We calculated the shortest time solution, the minimum angle solution, and the highest trajectory (safest) muzzle velocity when the target can be hit. The muzzle velocities are all 450m/s, with the highest muzzle angle being 75.8° and the lowest muzzle angle being 3.1°. When the lowest muzzle angle is 3.1°, the shortest time is 4.0s.

3. In response to the requirements of the problem, under real artillery firing conditions, while considering the energy loss (economic loss) caused by increased speed and the efficiency loss caused by extended time, we designed a normalized scoring function:

$$S_{\text{eco}} = (v_{\max} - v_{\text{self}})/(v_{\max} - v_{\min}) * 100$$

$$S_{\text{time}} = (t_{\max} - t_{\text{self}})/(t_{\max} - t_{\min}) * 100$$

$$S_{\text{final}} = 0.5 * S_{\text{eco}} + 0.5 * S_{\text{time}}$$

Where S_{eco} denotes Economic score at a given projectile muzzle velocity, $v_{\max}$ denotes the minimum value in the overall projectile muzzle velocity, $v_{\min}$ denotes the maximum value in the overall projectile muzzle velocity, v_{self} denotes current projectile muzzle velocity, so as the S_{time}. Finally, we weighted the sum of these two scores with equal weights of 0.5 each, yielding the S_{final} and calculated the scores for all solutions within the initial velocity range of 140-450 m/s, as shown in the figure4. Ultimately, we selected the scheme with an initial velocity of 165.0 m/s and a muzzle angle of 21.8°, which requires a flight time of approximately 11.41 seconds.

4. We can deduce from the diagram that for different muzzle velocities, two distinct muzzle angles can be achieved to hit the target, and the difference in muzzle angles increases with the increase in muzzle velocity. This provides some inspiration for our subsequent analysis to develop a simplified method for calculating artillery firing.

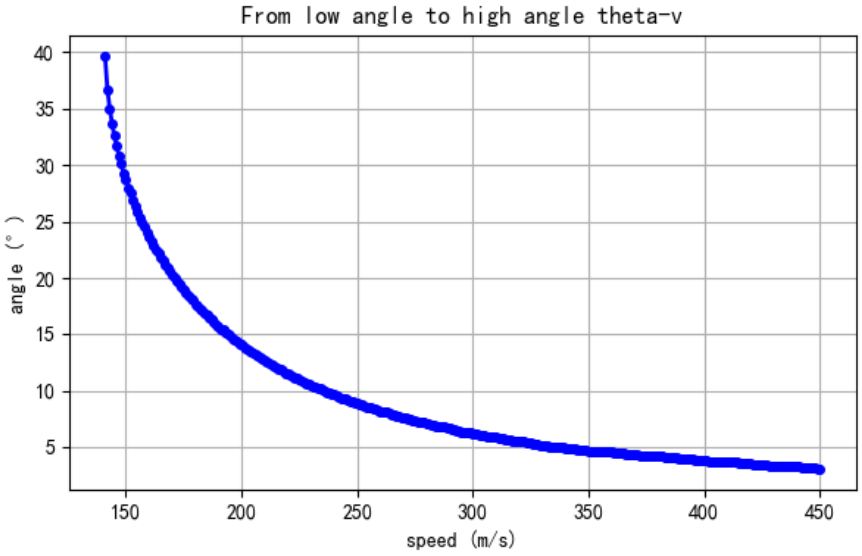

Figure 6: From low angle to high angle θ-v

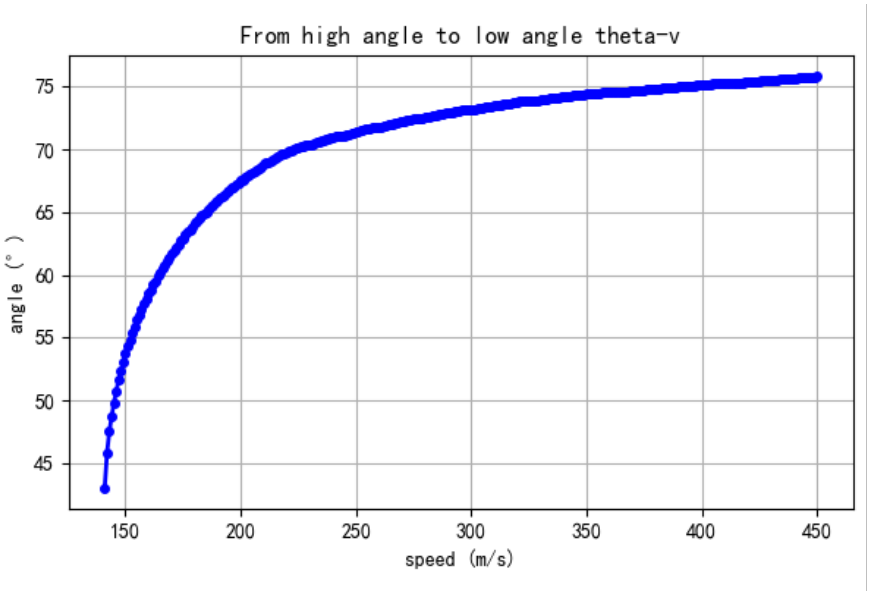

Figure 7: From high angle to low angle θ-v

speed	angle	time	economic score	time score	final score
165	21.8	10.88	92.23300971	73.70855821	82.97078396
166	21.5	10.8	91.90938511	74.01696222	82.96317367
172	19.7	10.3	89.96763754	75.94448728	82.95606241
167	21.2	10.72	91.58576052	74.32536623	82.95556337
170	20.3	10.47	90.61488673	75.28912876	82.95200775
168	20.9	10.64	91.26213592	74.63377024	82.94795308
171	20	10.39	90.29126214	75.59753277	82.94439745
169	20.6	10.56	90.93851133	74.94217425	82.94034279
175	18.9	10.07	88.99676375	76.8311488	82.91395628
163	22.5	11.08	92.8802589	72.93754819	82.90890354
174	19.2	10.16	89.32038835	76.48419429	82.90229132
164	22.2	11	92.5566343	73.2459522	82.90129325

Figure 8: The top scorers

3.2 Ballistic Analysis Under Multi-Target Distances and Environmental Factors

In this section, building upon the ballistic model established in Section 3.1, we extend our analysis to scenarios where target distances range from 1000 to 1500 meters, incorporating the combined effects of altitude variations and wind conditions. This expanded analysis aims to comprehensively evaluate the impact of these factors on ballistic trajectories and the resulting requirements for firing parameters. We enhance the model's realism by introducing a model for wind speed variation with altitude (WindBallistics class).

3.2.1 Effects of horizontal range and elevation variations between gun muzzle and target

To address variable target distances, we have enhanced the Python code presented in Section 3.1. The core motion equations remain unchanged, but when calculating target distances ranging from 1000 meters to 1500 meters, the search algorithm employed for each distance has been refined. Unlike the bidirectional search in Section 3.1, we have devised a penalty function to constrain elevation and azimuth angles, defined as follows:

$$E_{\text{hor}} = \sqrt{(impact_{\text{x}} - target_{\text{distance}})^2 + impact_{\text{x}}^2}$$

$$E_{\text{hei}} = |impact_{\text{z}} - (target_{\text{altitude}} - cannon_{\text{altitude}})|$$

$$E_{\text{total}} = weight_{\text{dist}} \times E_{\text{hor}} + weight_{\text{hei}} \times E_{\text{hei}}$$

$$\text{penalty} = \begin{cases} 1000 \times (0 - \text{elevation})^2, & \text{if elevation} < 0 \\ 1000 \times (\text{elevation} - 90)^2, & \text{if elevation} > 90 \\ 1000 \times (-180 - \text{azimuth})^2, & \text{if azimuth} < -180 \\ 1000 \times (\text{azimuth} - 180)^2, & \text{if azimuth} > 180 \\ 0, & \text{otherwise} \end{cases}$$

Based on experience, the initial elevation angle is set to 45°, while the initial azimuth angle is determined according to wind direction. The penalty function is then converged using either the SLSQP optimization method or a grid search approach to obtain suitable launch elevation

and azimuth angles. Furthermore, we incorporate altitude effects: employing the International Standard Atmosphere (ISA) model to compute temperature, pressure, and air density variations at different elevations. This adjusts the impact of altitude changes on drag (including both initial elevation and altitude variations during projectile motion), using a methodology analogous to Section 3.1. Here, the altitude h is adjusted based on the relative positions of the cannon and target point. This extension enables the model to handle scenarios such as firing from hills into valleys.

3.2.2 Influence of wind conditions

Wind conditions are a key factor affecting trajectory, though not discussed in Section 3.1. We treat the relative velocity between wind and projectile as a source of "drag":

$$F_{\text{drag},x} = -F_{\text{drag}}\frac{v_{\text{rel},x}}{v_{\text{rel}}}, \quad F_{\text{drag},y} = -F_{\text{drag}}\frac{v_{\text{rel},y}}{v_{\text{rel}}}, \quad F_{\text{drag},z} = -F_{\text{drag}}\frac{v_{\text{rel},z}}{v_{\text{rel}}}$$

Wind speed component (based on wind direction angle θ_w):

$$w_x = w(z)\cos(\theta_w), \quad w_y = w(z)\sin(\theta_w)$$

Where θ_w denotes wind direction (in degrees), with the downwind direction being 0°.

A logarithmic wind profile model, which accounts for wind speed variation with height, has been introduced. Specifically, the wind speed at absolute elevation z is calculated as follows:

$$w(z) = \begin{cases} 0 & \text{if } z \le z_0 \\ \dfrac{u_*}{0.4}\ln\left(\dfrac{z}{z_0}\right) & \text{if } z_0 < z \le h_c \\ w_c\left(1 + \dfrac{z - h_c}{h_{\text{ref}}}\right)^{\alpha} & \text{if } z > h_c \end{cases}$$

Where:

z denotes absolute elevation (unit: m), $z_0 = 0.1\,\text{m}$ represents surface roughness length, h_c is the cannon's elevation height, u_* is the friction velocity (w_c is the wind speed at the cannon location), $h_{\text{ref}} = \max(h_c, 10)$ is the reference height, and $\alpha = 0.143$ is the wind shear exponent.

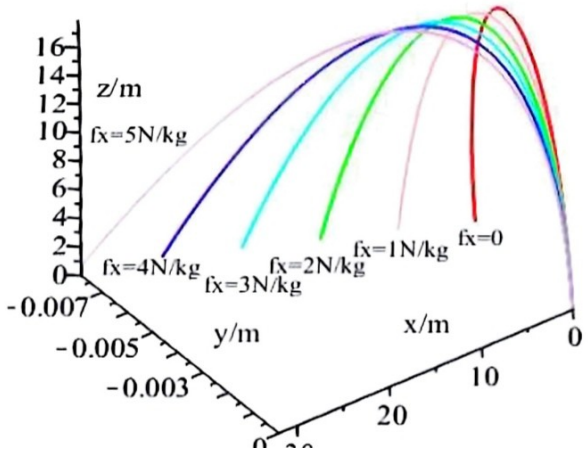

Figure 9: How different wind speeds affect projectile motion

This innovation enhances the model's physical realism, proving particularly effective in historical artillery scenarios with variable wind conditions. We implemented this extension via

Python code, simulating vertical variations in wind speed during ballistic calculations to achieve more accurate impact point predictions.

3.2.3 Presentation of Results and Trend Analysis

Through the designed program, we can input the target distance, altitudes of the two objects, wind direction and speed, as well as the launch speed, and then output the launch elevation angle, launch azimuth angle, and ballistic trajectory image that can hit the target. The 20 random input-output pairs are shown in Figure 10 below.

W_speed	W_direct	Distance	A_altitu	T_altitu	Speed	Elevatio	Azimuth
5.45	106.46	1037.88	670.11	878.03	335.13	15.41	−0.33
4.07	102.95	1272.53	750.15	961.44	257.85	18.34	−0.35
29.3	80.87	1139.75	733.85	501.95	444.46	77	−45
13.16	207.21	1463.29	578.17	583.98	339.12	7.51	0.61
4.29	162.9	1424.64	858.66	695.55	282.63	2.51	−0.13
9.56	359.7	1139.44	961.95	961.21	337.08	4.28	0
16.09	303.04	1486.85	873.74	656.84	368.61	0	1.13
14.31	201.46	1488.02	658.51	650.86	261.77	12.23	0.68
10	216.33	1149.58	942.11	913.85	357.93	2.9	0.41
21.59	350.41	1143.69	550.62	805.09	399.65	16.14	0.22
9.65	323.4	1487.36	522.46	563.44	314.47	9.29	0.55
6.55	92.84	1444.1	764.08	583.04	282.94	1.89	−0.67
9.13	236.72	1334	864.04	790.82	395.31	1.48	0.6
15.43	274.28	1461.05	802.67	957.15	221.79	22	5
0.55	62.51	1411.74	892.41	764.92	219.74	9.14	−0.06
11.18	196.22	1309.76	614.91	735.12	410.13	9.44	−0.01
6.29	104.69	1353.07	860.17	538.03	404.27	77	−5
19.73	332.2	1436.15	529.9	993.63	377.73	23.75	0.82
15.29	116.35	1021.31	740.32	679.64	242.06	4.09	−1.1
20.29	130.15	1245.12	599.98	892.06	391.42	17.54	−1.32

Figure 10: Random inspection of ballistic calculators

Meanwhile, we have provided several classic experimental examples below:

1. The wind speed is 0m/s, the wind direction is 0, the horizontal distance is 1200m, the artillery elevation is 500m, the target elevation is 500m, and the muzzle velocity is 141m/s.The result : Elevation angle is 43.59° and Azimuth angle is -0.00°.

2. The wind speed is 30m/s, the wind direction is 60 degrees, the horizontal distance is 1000m, the artillery elevation is 500m, the target elevation is 500m, and the muzzle velocity is 141m/s. The result : Elevation angle is 20.99° and Azimuth angle is -3.12° .

3. The wind speed is 30m/s, the wind direction is 90 degrees, the horizontal distance is 1000m, the artillery elevation is 500m, the target elevation is 500m, and the muzzle velocity is 400m/s. The result : Elevation angle is 2.89° and Azimuth angle is -1.74° .

4. The wind speed is 20m/s, the wind direction is 60 degrees, the horizontal distance is 1500m, the artillery elevation is 500m, the target elevation is 600m, and the muzzle velocity is 450m/s. The result : Elevation angle is 7.80° and Azimuth angle is -1.41° .

5. The wind speed is 20m/s, the wind direction is -60 degrees, the horizontal distance is 1000m, the artillery elevation is 500m, the target elevation is 600m, and the muzzle velocity is 450m/s. The result : Elevation angle is 8.02° and Azimuth angle is 0.90° .

In addition, we present the three-dimensional views of Example 3 and Example 5 as shown in Figures 11 and 12.

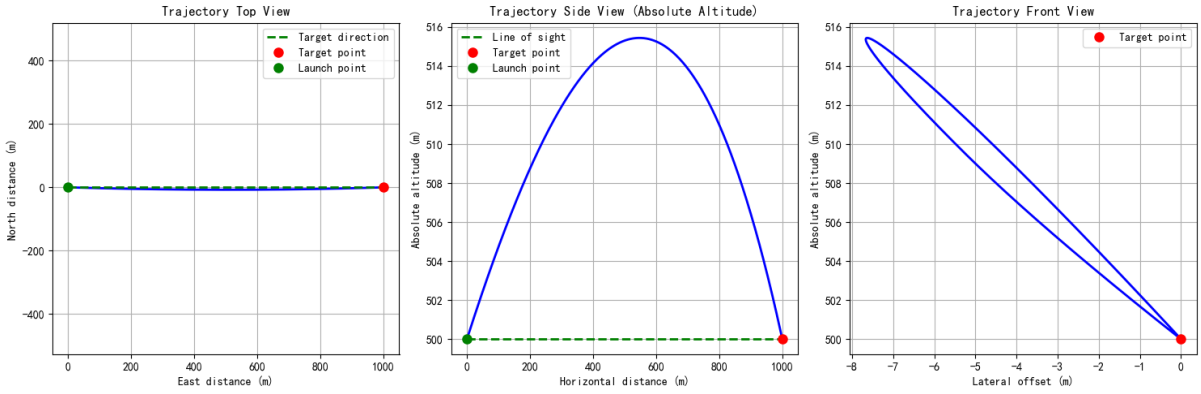

Figure 11: Trajectory TopView,SideView,FrontView of example 3

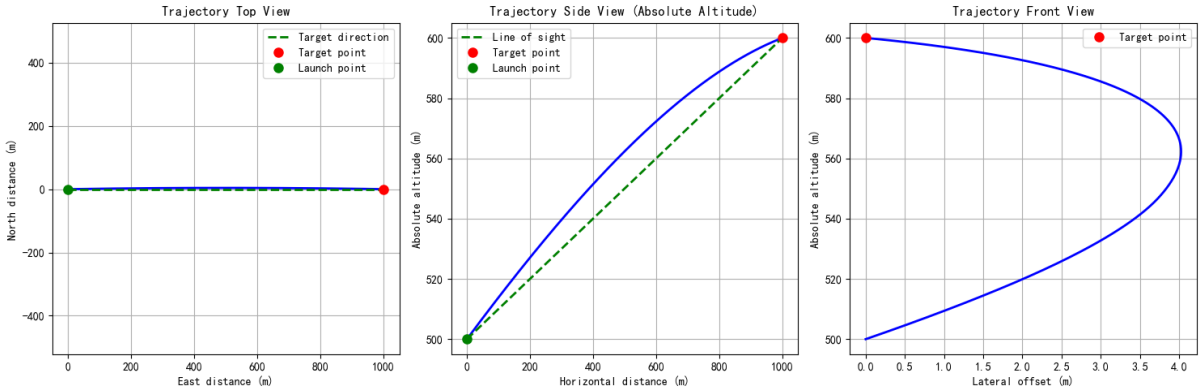

Figure 12: Trajectory TopView,SideView,FrontView of example 5

3.3 Design of rapid estimation method

3.3.1 Method 1: Function fitting method

The fitting function idea of this code is based on a method combining numerical optimization and regression analysis, aiming to establish a simplified mathematical relationship between launch parameters and ballistic conditions through limited sample data.

· Training point generation

Firstly, the precise elevation and azimuth angle of twenty randomly selected representative state points within the range are calculated through the ballistic physical model in Question 2. These state points cover the actual variation range of key variables such as wind speed, wind direction, target distance, and altitude. The calculation process takes into account complex physical factors such as air resistance, gravity changes, and wind speed profiles. Numerical integration and optimization algorithms are employed to ensure the accuracy of the results, providing reliable training data for fitting.

· Constructing a fitting model using multiple regression techniques

For the elevation, due to its nonlinear relationship with the input parameters, a quadratic polynomial regression model is adopted, with target distance, altitude difference, downwind component, and crosswind component as feature variables. The polynomial expansion is used to capture the interactions and higher-order effects between variables.

Elevation fitting formula (quadratic polynomial regression):

$$\text{Elevation} = \beta_0 + \sum_{i=1}^{4} \beta_i x_i + \sum_{i=1}^{4} \sum_{j=i}^{4} \beta_{ij} x_i x_j$$

Among them, the feature variable x_i includes: target distance, altitude difference (target altitude - muzzle altitude), downwind component (w_x), and crosswind component (w_y).

For the launch direction angle, due to its relatively linear variation, a linear regression model is adopted, with the same feature variables for fitting. The model training uses the least squares method to estimate parameters, aiming to minimize the sum of squared errors between the predicted values and the true values. Finally, a simplified calculation formula is obtained.

Direction angle fitting formula (linear regression):

$$\text{azimuth angle} = \gamma_0 + \sum_{i=1}^{4} \gamma_i x_i$$

Among them, the feature variable x_i is the same as the elevation fitting.

· Error analysis of fitting formula

The validation of the fitting model is conducted through an independent test set, with the evaluation metric being the mean absolute error. Here we give two combinations of datasets and test sets, namely: 20 datasets, 20 test sets, 100 datasets, and 100 test sets, and the results are shown in figure13 and 14. Obviously, the larger the dataset, the more accurate the emission angle given, so using the fitting result of the latter, the fitting function after substituting the coefficients is:

Elevation Angle Fitting Formula:

$$\begin{aligned} \text{Elevation} =& b_0 + a_0 + a_1 \times \text{distance} + a_2 \times \text{height}_{\text{diff}} + a_3 \times w_x + a_4 \times w_y \\ &+ a_5 \times \text{distance}^2 + a_6 \times \text{distance} \times \text{height}_{\text{diff}} + a_7 \times \text{distance} \times w_x \\ &+ a_8 \times \text{distance} \times w_y + a_9 \times \text{height}_{\text{diff}}^2 + a_{10} \times \text{height}_{\text{diff}} \times w_x \\ &+ a_{11} \times \text{height}_{\text{diff}} \times w_y + a_{12} \times w_x^2 + a_{13} \times w_x \times w_y + a_{14} \times w_y^2 \end{aligned} \tag{1}$$

Where a_i denotes distinct coefficients, h_{diff} denotes the elevation difference, w_x denotes the tailwind component, w_y denotes the crosswind component, and b_0 denotes intercept.

The specific values of the parameters are as follows: $a_0 = 0.033391$, $a_1 = 0.029002$, $a_2 = 0.299983$, $a_3 = 0.136904$, $a_4 = -0.000017$, $a_5 = -0.000014$, $a_6 = -0.000227$, $a_7 = -0.000150$, $a_8 = 0.000007$, $a_9 = 0.000005$, $a_{10} = -0.000074$, $a_{11} = -0.000339$, $a_{12} = -0.001170$, $a_{13} = -0.001735$, $b_0 = -5.556424$.

Azimuth Fitting Formula:

$$\text{Azimuth} = b_0 + b_1 \times \text{distance} + b_3 \times w_x + b_4 \times w_y \tag{2}$$

The specific values of the parameters are as follows: $b_0 = 0.543220$, $b_1 = -0.000481$, $b_3 = 0.005298$, $b_4 = -0.071935$.

At this moment, the relative error of the pitch angle is approximately:ϵ=2.44/20. 1=12.2%; The relative error of the azimuth is approximately: ϵ=0.46/21=21.9%. The combined error of the two, after averaging, is: ϵ=17.1%.

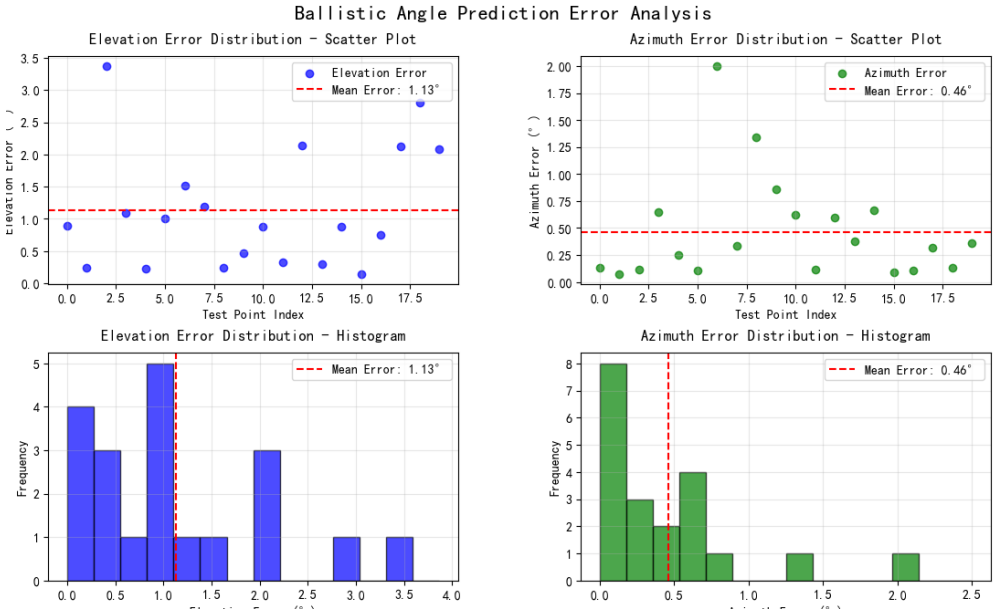

Figure 13: Error analysis 1

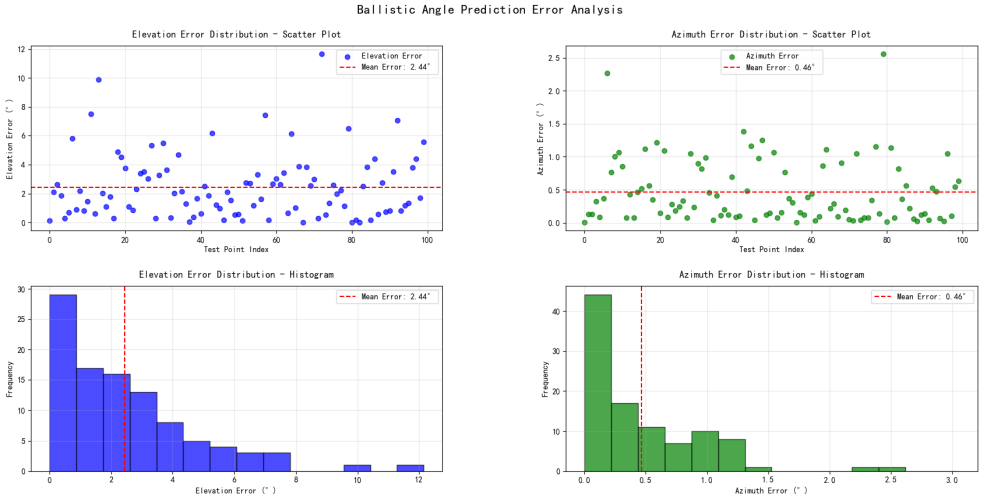

Figure 14: Error analysis 2

3.3.2 Method 2: Table lookup method

To enhance the speed of command for officers and soldiers, we have constructed a simple correspondence table. We have set the muzzle velocity at 450m/s, while selecting wind speeds of 0, 10, 20, and 30, wind directions of 0, 90, 180, and 270, distances of 1000, 1200, and 1500, and altitudes of the artillery and the target at 500, 750, and 1000 for combination. This results in a

table with 4*4*3*3*3 rows, with each row corresponding to a relatively accurate elevation and positioning angle.Figure 12 is a partial extraction of the shooting table with a step size of 10. It should be noted that due to the discrete nature of the table, elevation and azimuth values for certain data conditions may not be explicitly listed. However, we can make rough estimations within specified data ranges to determine the required elevation and azimuth angles.

W_speed(m/s)	W_direction(°)	Distance(m)	Altitude_A(m)	Altitude_T(m)	S_speed(m/s)	Elevation(°)	Azimuth(°
0	0	1000	500	500	450	2.3	0
0	0	1200	500	750	450	14.9	0
0	0	1500	500	1000	450	23.2	0
0	90	1000	750	750	450	2.28	0
0	90	1200	750	1000	450	14.85	0
0	90	1500	1000	1000	450	4.3	0
0	180	1000	500	500	450	2.3	0
0	180	1200	500	750	450	14.9	0
0	180	1500	500	1000	450	23.2	0
0	270	1000	750	750	450	2.28	0
0	270	1200	750	1000	450	14.85	0
0	270	1500	1000	1000	450	4.3	0
10	0	1000	500	500	450	2.25	0
10	0	1200	500	750	450	14.92	0
10	0	1500	500	1000	450	23.25	0
10	90	1000	750	750	450	2.28	-0.52
10	90	1200	750	1000	450	14.85	-0.67
10	90	1500	1000	1000	450	4.3	-0.8
10	180	1000	500	500	450	77	0
10	180	1200	500	750	450	14.87	0
10	180	1500	500	1000	450	23.16	0
10	270	1000	750	750	450	2.28	0.52
10	270	1200	750	1000	450	14.85	0.67
10	270	1500	1000	1000	450	4.3	0.8
20	0	1000	500	500	450	2.19	0
20	0	1200	500	750	450	14.95	0
20	0	1500	500	1000	450	23.29	0
20	90	1000	750	750	450	2.28	-1.05
20	90	1200	750	1000	450	14.86	-1.34
20	90	1500	1000	1000	450	4.31	-1.61
20	180	1000	500	500	450	2.44	-1.55
20	180	1200	500	750	450	14.85	0
20	180	1500	500	1000	450	23.13	0
20	270	1000	750	1000	450	16.37	1.11
20	270	1200	750	750	450	3.02	1.28
20	270	1500	1000	1000	450	4.31	1.61
30	0	1000	500	500	450	2.14	0
30	0	1200	500	750	450	14.97	0
30	0	1500	500	1000	450	23.34	0
30	90	1000	750	1000	450	16.37	-1.68
30	90	1200	1000	1000	450	2.98	-1.73
30	90	1500	500	500	450	4.48	-2.56
30	180	1000	500	750	450	16.17	0
30	180	1200	500	1000	450	25.4	0
30	180	1500	750	750	450	65	0
30	270	1000	750	1000	450	16.37	1.68
30	270	1200	1000	1000	450	2.98	1.87
30	270	1500	750	750	450	4.4	2.49

Figure 15: Firing table

4 Model Evaluation

4.1 Strengths

Our advantages are as follows:

1.In the model used to solve the first question, we did not simply treat air resistance as a variable solely related to velocity. Instead, we considered the changes in atmospheric pressure and density due to altitude effects during the flight of the cannonball, which in turn led to changes in the Mach number of the cannonball. Ultimately, the resistance experienced by the

cannonball during flight not only varies with velocity but also with altitude. This model takes this into account and consistently adheres to this basic principle in subsequent models, making the model more practical.

2.In the model used to solve the second question, we decomposed the impact of air volume in different directions, allowing the calculations from the previous model to be largely reused. Furthermore, we conducted a more detailed calculation of the impact of air volume, as we simultaneously considered changes in wind speed when altitude changes, making the overall model calculation more precise.

3. The fitting function of the third question is highly operable, as it can accurately obtain two launch angles by inputting six known variables from the battlefield. The core advantage of this method lies in transforming complex ballistic calculation problems into simple algebraic operations, significantly enhancing computational efficiency while maintaining high prediction accuracy, thus providing a practical solution for real-time fire control systems. The entire process embodies the effective integration of physical models and data-driven methods, ensuring that the model retains physical significance while being computationally simple.It is comprehensive and rigorous.

4. Finally, we developed a function fitting method and a table lookup method, allowing artillery officers who have received mathematical training but do not have access to computers or calculators to choose between more accurate or faster estimations.

4.2 Weaknesses

Our disadvantages are as follows:

1.Regarding the model designed for the first question, our approach is relatively comprehensive. As the first question did not require consideration of airflow effects, we did not factor in wind speed. However, one potential imperfection in the model designed for the first question is its inability to precisely determine the minimum initial velocity required to hit the target. This is because our bidirectional search algorithm yields the first velocity that strikes the target's right side, resulting in the searched minimum initial velocity being larger than the true minimum velocity.

2.The model designed for the second question exhibits the following weaknesses: · Due to the use of a hierarchical optimisation approximation algorithm, for each given velocity we can only provide one corresponding angle capable of hitting the target. This angle may be either too large or too small. As the optimisation algorithm imposes relatively minor constraints on penalties, this may result in solutions of insufficient precision. · Although the drag model accounts for wind speed variations with altitude and the effects of atmospheric pressure and density changes on the drag coefficient, it may still exhibit some inaccuracies compared to real-world scenarios. Employing the G1/G7 standard could yield a more precise drag model.

3.The model designed for the third question exhibits the following limitations: · Regarding the determination of an appropriate muzzle velocity mentioned in the third question, we have fixed this at 450 m/s within the model. This is because, in practical artillery operation, muzzle velocity is difficult to adjust, whereas elevation angle and azimuth angle remain more readily controllable. Therefore, we have fixed the muzzle velocity in the model. Should data for different velocities be required, the ballistic calculator provided by the second model can be utilised for calculations. · The mean error data for the fitted elevation angle and azimuth angle formulas in our model are derived from the full range of all data points. This approach may result in suboptimal fitting performance within specific local data regions. To improve this, separate fitting could be performed on different local data segments to better accommodate the requirements of each distinct data range.

5 Conclusions

Our solution systematically addresses the three challenges presented in Problem B. We first established a high-fidelity physical model, then extended it to complex conditions, and finally developed practical methods for manual calculation.

1. Core Calculation for 1200m Target

· Question: For a target at 1200m with both cannon and target at 500m altitude, find the muzzle velocity and firing angle.

· Solution & Result: We developed a ballistic model incorporating air resistance (with altitude-dependent density and Mach-number-dependent drag coefficient), and gravity variation. Using a bidirectional search algorithm (v: 100-450 m/s, θ: θ-90°), we found that multiple (v, θ) pairs can hit the target. The most energy-efficient solution is:

Minimum Muzzle Velocity : 141.0 m/s

Corresponding Firing Angle : 39.6°

Flight Time : 15.73 s

Considering a balance between energy consumption and flight time, we selected an optimal solution with a higher score:

Recommended Muzzle Velocity : 165.0 m/s

Recommended Firing Angle : 21.8°

The relationship between muzzle velocity and the required firing angle is non-linear and bimodal, as clearly shown in our figure results.

2. Extended Analysis for Variable Conditions

· Question: Extend the analysis to ranges of 1000-1500m, various altitudes, and wind conditions.

· Solution & Result: We enhanced our model by implementing a logarithmic wind profile to simulate wind speed variation with altitude and incorporating its vector into the force calculations. A hierarchical optimization algorithm was designed to automatically find the optimal (elevation, azimuth) angles that minimize the impact error for any given set of conditions (wind speed/direction, distance, altitudes, muzzle velocity).

Example Output :

Input : Wind Speed=20m/s, Wind Direction=60°, Distance=1000m, Cannon Alt=500m, Target Alt=600m, Muzzle Velocity=450m/s

Output : Elevation 7.80°, Azimuth -1.41° (a negative value indicates adjustment to the right of the target direction)

This enhanced model serves as the core for our subsequent rapid estimation calculator.

3. Development of Rapid Estimation Method

· Question: Develop a method for an artillery officer to quickly estimate parameters without computers.

· Solution & Result: We developed two distinct practical methods:

a) Function Calculation Method :

We fitted a multivariate quadratic function to the data generated by our high-fidelity model. The officer can compute the angles using:

$$\text{Elevation} = \beta_0 + \sum_{i=1}^{4} \beta_i x_i + \sum_{i=1}^{4} \sum_{j=i}^{4} \beta_{ij} x_i x_j$$

$$\text{azimuth angle} = \gamma_0 + \sum_{i=1}^{4} \gamma_i x_i$$

Where variables x_i represent target distance, altitude difference, downwind, and crosswind components. This method offers higher accuracy.

b) Table Lookup Method :

We pre-calculated a comprehensive firing table for discrete values of key parameters (e.g., distance: 1000, 1200, 1500m; wind speed: 0, 10, 20, 30 m/s). The officer simply finds the closest match to the current conditions and reads the corresponding elevation and azimuth angles. This method is faster and simpler.

Both methods are designed to be executed with only pen, paper, and basic mathematical knowledge, perfectly fulfilling the requirement for historical practicality.

In summary, this project provides a complete workflow from physical modeling and numerical simulation to the delivery of practical tools, offering accurate and historically plausible solutions to the artillery targeting problems posed in the competition.

References

[1] Dimitrios Gkritzapis et al. Computational Atmospheric Trajectory Simulation Analysis of Spin-Stabilized Projectiles and Small Bullets[J] International Journal of Computing Science and Mathematics, 2008.1

[2] Jiaqi Yao et al. Ballistic Object Trajectory Separation Point Estimation Using Fourth Order Runge-Kutta Method and Extended Kalman Filter[J]. 17th International Congress on Image and Signal Processing, BioMedical Engineering and Informatic, 2024

Appendix A Python source code

```
import numpy as np
import math
import matplotlib.pyplot as plt
import matplotlib as mpl

plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei', 'DejaVu Sans']
plt.rcParams['axes.unicode_minus'] = False

class CannonBallistics:
def __init__(self):
self.diameter = 0.11
self.mass = 5.0
self.radius = self.diameter / 2
self.g = 9.80665
self.altitude = 500
self.target_range = 1200
self.T0 = 288.15
self.P0 = 101325.0
self.rho0 = 1.225
self.L = 0.0065
```

```
self.R = 287.05
self.gamma = 1.4
self.A = math.pi * (self.radius) ** 2

def gravity_at_height(self, height):
R_earth = 6371000  # Earth radius (m)
return self.g * (R_earth / (R_earth + self.altitude + height)) ** 2

def atmosphere_at_height(self, height):
absolute_height = self.altitude + height
if absolute_height <= 11000:
T = self.T0 - self.L * absolute_height
exponent = self.g * 0.0289644 / (8.31432 * self.L)
P = self.P0 * (T / self.T0) ** exponent
rho = self.rho0 * (T / self.T0) ** (exponent - 1)
v_sound = math.sqrt(self.gamma * self.R * T)
else:
T = 216.65
P = 22632.0
rho = 0.3639
v_sound = math.sqrt(self.gamma * self.R * T)
return T, P, rho, v_sound

def drag_coefficient(self, mach):
if mach < 0.3:
return 0.45
elif mach < 0.8:
return 0.45 + 0.1 * (mach - 0.3)
elif mach < 1.0:
return 0.5 + 0.2 * (mach - 0.8)
elif mach < 1.2:
return 0.7 - 0.1 * (mach - 1.0)
else:
return 0.6

def air_resistance(self, velocity, height):
T, P, rho, v_sound = self.atmosphere_at_height(height)
if v_sound > 0:
mach = velocity / v_sound
else:
mach = 0
Cd = self.drag_coefficient(mach)
F_drag = 0.5 * rho * Cd * self.A * velocity ** 2
return F_drag

def acceleration_components(self, vx, vy, x, y):
v = math.sqrt(vx ** 2 + vy ** 2)

if v < 1e-6:
g_local = self.gravity_at_height(y)
return 0, -g_local

g_local = self.gravity_at_height(y)
F_drag = self.air_resistance(v, y)
```

```
a_drag = F_drag / self.mass

ax = -a_drag * (vx / v)
ay = -g_local - a_drag * (vy / v)

return ax, ay

def simulate_trajectory(self, v0, theta_deg, dt=0.01, max_time=77):
theta = math.radians(theta_deg)
x, y = 0, 0
vx = v0 * math.cos(theta)
vy = v0 * math.sin(theta)
trajectory = [(x, y)]
times = [0]
t = 0
while t < max_time and y >= -0.1:
ax, ay = self.acceleration_components(vx, vy, x, y)

vx += ax * dt
vy += ay * dt

x += vx * dt
y += vy * dt

trajectory.append((x, y))
t += dt
times.append(t)

if x > self.target_range * 2 and y < -10:
break

return np.array(trajectory), np.array(times)

def calculate_range_and_time(self, v0, theta_deg):
trajectory, times = self.simulate_trajectory(v0, theta_deg)
if len(trajectory) < 2:
return 0, 0
for i in range(1, len(trajectory)):
if trajectory[i, 1] <= 0:
x1, y1 = trajectory[i - 1]
x2, y2 = trajectory[i]

if y2 != y1:
t = -y1 / (y2 - y1)
impact_x = x1 + t * (x2 - x1)
impact_time = times[i - 1] + t * (times[i] - times[i - 1])
return impact_x, impact_time

return trajectory[-1, 0], times[-1]  # If the projectile doesn't hit the
    ground, return last position and time

def search_low_to_high(self):
print("Starting search from low to high...")
print("Velocity range: 100-450 m/s, Angle range: 0-90°")
```

```
print("=" * 50)

results = []
v0 = 100.0
v_step = 1.0
theta_step = 0.1

while v0 <= 450.0:
theta = 0.0

while theta <= 90.0:
range_val, flight_time = self.calculate_range_and_time(v0, theta)

if range_val > self.target_range:
results.append((v0, theta, range_val, flight_time))
print(f"Velocity {v0:.1f} m/s, Angle {theta:.1f}°, Range {range_val:.1f}
    m, Flight time {flight_time:.2f} s")
break  # Found first angle that exceeds 1200m, break out of angle loop
theta += theta_step

v0 += v_step

return results

def search_high_to_low(self):
print("\nStarting search from high to low...")
print("Velocity range: 450-100 m/s, Angle range: 90-0°")
print("=" * 50)

results = []
v0 = 450.0
v_step = -1.0
theta_step = -0.1

while v0 >= 100.0:
theta = 90.0

while theta >= 0.0:
range_val, flight_time = self.calculate_range_and_time(v0, theta)

if range_val > self.target_range:
results.append((v0, theta, range_val, flight_time))
print(f"Velocity {v0:.1f} m/s, Angle {theta:.1f}°, Range {range_val:.1f}
    m, Flight time {flight_time:.2f} s")
break

theta += theta_step

v0 += v_step

return results

def plot_angle_vs_velocity(self, results_low_high, results_high_low):
if results_low_high:
```

```
v_low_high = [r[0] for r in results_low_high]
theta_low_high = [r[1] for r in results_low_high]
flight_time_low_high = [r[3] for r in results_low_high]

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(v_low_high, theta_low_high, 'bo-', linewidth=2, markersize=4)
plt.xlabel('Speed (m/s)')
plt.ylabel('Angle (°)')
plt.title('From low angle to high angle theta-v')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(v_low_high, flight_time_low_high, 'go-', linewidth=2, markersize
    =4)
plt.xlabel('Speed (m/s)')
plt.ylabel('Flight time (s)')
plt.title('From low angle to high angle t-v')
plt.grid(True)

plt.tight_layout()
plt.show()

if results_high_low:
v_high_low = [r[0] for r in results_high_low]
theta_high_low = [r[1] for r in results_high_low]
flight_time_high_low = [r[3] for r in results_high_low]

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(v_high_low, theta_high_low, 'bo-', linewidth=2, markersize=4)
plt.xlabel('Speed (m/s)')
plt.ylabel('Angle (°)')
plt.title('From high angle to low angle theta-v')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(v_high_low, flight_time_high_low, 'go-', linewidth=2, markersize
    =4)
plt.xlabel('Speed (m/s)')
plt.ylabel('Flight time (s)')
plt.title('From high angle to low angle t-v')
plt.grid(True)

plt.tight_layout()
plt.show()

def analyze_optimal_solution(self, results):
if not results:
print("No solution found that meets the conditions")
return

```

```
best_solution = min(results, key=lambda x: abs(x[2] - self.target_range))
v0, theta, range_val, flight_time = best_solution

print("\n" + "=" * 50)
print("Optimal solution analysis:")
print(f"Velocity: {v0:.1f} m/s")
print(f"Angle: {theta:.1f}°")
print(f"Range: {range_val:.1f} m")
print(f"Flight time: {flight_time:.2f} s")
print(f"Error: {abs(range_val - self.target_range):.1f} m")

trajectory, times = self.simulate_trajectory(v0, theta)
max_height = np.max(trajectory[:, 1])

print(f"Maximum height: {max_height:.1f} m")

vacuum_range = (v0 ** 2 * math.sin(2 * math.radians(theta))) / self.g
vacuum_time = (2 * v0 * math.sin(math.radians(theta))) / self.g
print(f"Vacuum trajectory range: {vacuum_range:.1f} m")
print(f"Vacuum trajectory flight time: {vacuum_time:.2f} s")
print(f"Range reduction due to air resistance: {vacuum_range - range_val
    :.1f} m")
print(f"Flight time increase due to air resistance: {flight_time -
    vacuum_time:.2f} s")


def main():
print("Cannon Ballistics Calculator - Bidirectional Search and
    Visualization")
print("=" * 60)

ballistics = CannonBallistics()

print(f"Target distance: {ballistics.target_range} m")
print(f"Launch altitude: {ballistics.altitude} m")
print()

results_low_high = ballistics.search_low_to_high()

results_high_low = ballistics.search_high_to_low()

ballistics.plot_angle_vs_velocity(results_low_high, results_high_low)

all_results = results_low_high + results_high_low
ballistics.analyze_optimal_solution(all_results)


if __name__ == "__main__":
main()
```

```
import numpy as np
import math
from scipy.optimize import minimize
```

```
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import TextBox
import matplotlib
matplotlib.use('TkAgg')
plt.rcParams['font.sans-serif'] = ['SimHei', 'Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False
matplotlib.use('TkAgg')


class WindBallistics:
def __init__(self):
self.diameter = 0.11
self.mass = 5.0
self.radius = self.diameter / 2
self.g0 = 9.80665
self.R_earth = 6371000
self.T0 = 288.15
self.P0 = 101325.0
self.rho0 = 1.225
self.L = 0.0065
self.R_air = 287.05
self.gamma = 1.4
self.A = math.pi * (self.radius) ** 2
self.wind_speed = 0.0
self.wind_direction = 0.0
self.target_distance = 1000.0
self.cannon_altitude = 0.0
self.target_altitude = 0.0
self.muzzle_velocity = 300.0

def set_parameters(self, wind_speed, wind_direction, target_distance,
cannon_altitude, target_altitude, muzzle_velocity):
self.wind_speed = wind_speed
self.wind_direction = wind_direction
self.target_distance = target_distance
self.cannon_altitude = cannon_altitude
self.target_altitude = target_altitude
self.muzzle_velocity = muzzle_velocity

def gravity_at_height(self, absolute_height):
return self.g0 * (self.R_earth / (self.R_earth + absolute_height)) ** 2

def atmosphere_at_height(self, absolute_height):
if absolute_height <= 11000:
T = self.T0 - self.L * absolute_height
exponent = self.g0 * 0.0289644 / (8.31432 * self.L)
P = self.P0 * (T / self.T0) ** exponent
rho = self.rho0 * (T / self.T0) ** (exponent - 1)
v_sound = math.sqrt(self.gamma * self.R_air * T)
else:
T = 216.65
```

```
P = 22632.0
rho = 0.3639
v_sound = math.sqrt(self.gamma * self.R_air * T)

return T, P, rho, v_sound

def drag_coefficient(self, mach):
if mach < 0.3:
return 0.45
elif mach < 0.8:
return 0.45 + 0.1 * (mach - 0.3)
elif mach < 1.0:
return 0.5 + 0.2 * (mach - 0.8)
elif mach < 1.2:
return 0.7 - 0.1 * (mach - 1.0)
else:
return 0.6

def wind_components(self):
wind_dir_rad = math.radians(self.wind_direction)
wind_x = self.wind_speed * math.cos(wind_dir_rad)
wind_y = self.wind_speed * math.sin(wind_dir_rad)
return wind_x, wind_y

def air_resistance(self, vx, vy, vz, absolute_height):
T, P, rho, v_sound = self.atmosphere_at_height(absolute_height)
wind_x, wind_y = self.wind_components()
v_rel_x = vx - wind_x
v_rel_y = vy - wind_y
v_rel_z = vz
v_rel = math.sqrt(v_rel_x ** 2 + v_rel_y ** 2 + v_rel_z ** 2)

if v_sound > 0:
mach = v_rel / v_sound
else:
mach = 0

Cd = self.drag_coefficient(mach)
F_drag = 0.5 * rho * Cd * self.A * v_rel ** 2

if v_rel > 1e-6:
F_drag_x = -F_drag * (v_rel_x / v_rel)
F_drag_y = -F_drag * (v_rel_y / v_rel)
F_drag_z = -F_drag * (v_rel_z / v_rel)
else:
F_drag_x, F_drag_y, F_drag_z = 0, 0, 0

return F_drag_x, F_drag_y, F_drag_z

def acceleration_components(self, vx, vy, vz, x, y, absolute_height):
g_local = self.gravity_at_height(absolute_height)
F_drag_x, F_drag_y, F_drag_z = self.air_resistance(vx, vy, vz,
    absolute_height)
ax = F_drag_x / self.mass
```

```
ay = F_drag_y / self.mass
az = -g_local + F_drag_z / self.mass
return ax, ay, az

def simulate_trajectory(self, elevation_deg, azimuth_deg, dt=0.01,
    max_time=60):
elevation = math.radians(elevation_deg)
azimuth = math.radians(azimuth_deg)
v0 = self.muzzle_velocity

#    -
x, y, z_abs = 0, 0, self.cannon_altitude  #    z_abs
vx = v0 * math.cos(elevation) * math.cos(azimuth)
vy = v0 * math.cos(elevation) * math.sin(azimuth)
vz = v0 * math.sin(elevation)

state = np.array([x, y, z_abs, vx, vy, vz])
trajectory = [state[:3].copy()]
velocities = [state[3:].copy()]

t = 0
target_reached = False


while t < max_time and z_abs >= -100:

if x >= self.target_distance and not target_reached:
target_reached = True

if len(trajectory) > 1:
x_prev, y_prev, z_prev = trajectory[-2]
x_curr, y_curr, z_curr = trajectory[-1]

if x_curr != x_prev:
t_interp = (self.target_distance - x_prev) / (x_curr - x_prev)
x_interp = self.target_distance
y_interp = y_prev + t_interp * (y_curr - y_prev)
z_interp = z_prev + t_interp * (z_curr - z_prev)
trajectory[-1] = (x_interp, y_interp, z_interp)
state[:3] = [x_interp, y_interp, z_interp]
break

#
ax, ay, az = self.acceleration_components(vx, vy, vz, x, y, z_abs)

#     -
k1 = np.array([vx, vy, vz, ax, ay, az])
k2_state = state + 0.5 * dt * k1
k2_vx, k2_vy, k2_vz = k2_state[3], k2_state[4], k2_state[5]
k2_ax, k2_ay, k2_az = self.acceleration_components(k2_vx, k2_vy, k2_vz,
    k2_state[0], k2_state[1],
k2_state[2])
k2 = np.array([k2_vx, k2_vy, k2_vz, k2_ax, k2_ay, k2_az])

```

```
k3_state = state + 0.5 * dt * k2
k3_vx, k3_vy, k3_vz = k3_state[3], k3_state[4], k3_state[5]
k3_ax, k3_ay, k3_az = self.acceleration_components(k3_vx, k3_vy, k3_vz,
    k3_state[0], k3_state[1],
k3_state[2])
k3 = np.array([k3_vx, k3_vy, k3_vz, k3_ax, k3_ay, k3_az])

k4_state = state + dt * k3
k4_vx, k4_vy, k4_vz = k4_state[3], k4_state[4], k4_state[5]
k4_ax, k4_ay, k4_az = self.acceleration_components(k4_vx, k4_vy, k4_vz,
    k4_state[0], k4_state[1],
k4_state[2])
k4 = np.array([k4_vx, k4_vy, k4_vz, k4_ax, k4_ay, k4_az])

state += (dt / 6.0) * (k1 + 2 * k2 + 2 * k3 + k4)
x, y, z_abs, vx, vy, vz = state

trajectory.append(state[:3].copy())
velocities.append(state[3:].copy())
t += dt

#
if x > self.target_distance * 2 and z_abs < self.target_altitude - 100:
break

#
if z_abs > 10000 and x > self.target_distance:
break

return np.array(trajectory), np.array(velocities)

def calculate_impact_point(self, elevation_deg, azimuth_deg):
trajectory, velocities = self.simulate_trajectory(elevation_deg,
    azimuth_deg)

if len(trajectory) < 2:
return float('inf'), float('inf'), float('inf')

for i in range(len(trajectory) - 1, 0, -1):
z_current = trajectory[i, 2]  #
z_prev = trajectory[i - 1, 2]

if (z_prev >= self.target_altitude and z_current <= self.target_altitude)
     or \
(z_prev <= self.target_altitude and z_current >= self.target_altitude):

if z_current != z_prev:
t = (self.target_altitude - z_prev) / (z_current - z_prev)
impact_x = trajectory[i - 1, 0] + t * (trajectory[i, 0] - trajectory[i -
    1, 0])
impact_y = trajectory[i - 1, 1] + t * (trajectory[i, 1] - trajectory[i -
    1, 1])
impact_z = self.target_altitude
return impact_x, impact_y, impact_z
```

```
211
212 return trajectory[-1, 0], trajectory[-1, 1], trajectory[-1, 2]
213
214 def objective_function(self, params):
215 elevation, azimuth = params
216 penalty = 0
217 if elevation < 0:
218 penalty += 1000 * (0 - elevation) ** 2
219 if elevation > 90:
220 penalty += 1000 * (elevation - 90) ** 2
221 if azimuth < -180:
222 penalty += 1000 * (-180 - azimuth) ** 2
223 if azimuth > 180:
224 penalty += 1000 * (azimuth - 180) ** 2
225
226 impact_x, impact_y, impact_z = self.calculate_impact_point(elevation,
    azimuth)
227 horizontal_error = math.sqrt((impact_x - self.target_distance) ** 2 +
    impact_y ** 2)
228 height_error = abs(impact_z - self.target_altitude)
229 distance_weight = 1.0
230 height_weight = max(0.5, 10.0 / max(1.0, self.target_distance))
231 total_error = distance_weight * horizontal_error + height_weight *
    height_error
232 return total_error + penalty
233
234 def find_optimal_angles(self):
235 g_effective = self.gravity_at_height(self.cannon_altitude)
236 R = self.target_distance
237 v0 = self.muzzle_velocity
238 height_diff = self.target_altitude - self.cannon_altitude
239
240 if v0 ** 2 > g_effective * R:
241 sin_2theta = g_effective * R / v0 ** 2
242 if sin_2theta <= 1.0:
243 initial_elevation = 0.5 * math.asin(sin_2theta)
244 initial_elevation = math.degrees(initial_elevation)
245 else:
246 initial_elevation = 45.0
247 else:
248 initial_elevation = 45.0
249
250 #
251 if abs(height_diff) > 0.1:
252 angle_correction = math.atan(height_diff / R)
253 initial_elevation += math.degrees(angle_correction) * 0.3
254
255 wind_x, wind_y = self.wind_components()
256 if abs(wind_x) > 0.1 or abs(wind_y) > 0.1:
257 wind_effect = math.sqrt(wind_x ** 2 + wind_y ** 2) / v0
258 initial_azimuth = -math.degrees(math.atan2(wind_y, wind_x)) * wind_effect
259 else:
260 initial_azimuth = 0.0
261
```

```
bounds = [(max(0, initial_elevation - 30), min(90, initial_elevation + 30)),
(max(-45, initial_azimuth - 45), min(45, initial_azimuth + 45))]

methods = ['SLSQP', 'L-BFGS-B']
best_result = None

for method in methods:
try:
result = minimize(self.objective_function,
[initial_elevation, initial_azimuth],
method=method,
bounds=bounds,
options={'ftol': 1e-6, 'maxiter': 200})
if result.success and (best_result is None or result.fun < best_result.fun):
best_result = result
except:
continue

if best_result and best_result.fun < 50:
return best_result.x[0], best_result.x[1], best_result.fun
else:
return self.refined_grid_search()

def refined_grid_search(self, elevation_step=2, azimuth_step=5):
best_error = float('inf')
best_elevation = 45
best_azimuth = 0

for elevation in range(10, 80, elevation_step * 2):
for azimuth in range(-45, 46, azimuth_step * 2):
error = self.objective_function([elevation, azimuth])
if error < best_error:
best_error = error
best_elevation = elevation
best_azimuth = azimuth

for elevation in range(max(0, best_elevation - 5), min(90, best_elevation + 6), elevation_step):
for azimuth in range(max(-90, best_azimuth - 10), min(91, best_azimuth + 11), azimuth_step):
error = self.objective_function([elevation, azimuth])
if error < best_error:
best_error = error
best_elevation = elevation
best_azimuth = azimuth

return best_elevation, best_azimuth, best_error

def calculate_solution(self):
print("Starting calculation of firing parameters...")
print("=" * 50)
print(f"Wind speed: {self.wind_speed} m/s")
```

```
print(
f"Wind direction: {self.wind_direction}° (0=headwind, 90=crosswind from left, 180=tailwind, 270=crosswind from right)")
print(f"Target distance: {self.target_distance} m")
print(f"Cannon altitude: {self.cannon_altitude} m")
print(f"Target altitude: {self.target_altitude} m")
print(f"Muzzle velocity: {self.muzzle_velocity} m/s")
print()

if self.target_altitude > self.cannon_altitude:
min_elevation = math.degrees(
math.asin((self.target_altitude - self.cannon_altitude) / self.target_distance))
print(f"Note: Target is {self.target_altitude - self.cannon_altitude:.1f} m above cannon")
print(f"Theoretical minimum elevation: {min_elevation:.1f}°")

elevation, azimuth, error = self.find_optimal_angles()
impact_x, impact_y, impact_z = self.calculate_impact_point(elevation, azimuth)
horizontal_error = math.sqrt((impact_x - self.target_distance) ** 2 + impact_y ** 2)
height_error = abs(impact_z - self.target_altitude)

print("\nCalculation results:")
print("=" * 50)
print(f"Elevation angle: {elevation:.2f}°")
print(f"Azimuth angle: {azimuth:.2f}° (relative to target direction)")
print(f"Predicted impact point: X={impact_x:.1f}m, Y={impact_y:.1f}m, Z={impact_z:.1f}m")
print(f"Horizontal error: {horizontal_error:.2f} m")
print(f"Altitude error: {height_error:.2f} m")
print(f"Total error: {error:.2f} m")

trajectory, velocities = self.simulate_trajectory(elevation, azimuth)
flight_time = len(trajectory) * 0.01
max_height = np.max(trajectory[:, 2]) if len(trajectory) > 0 else 0

print(f"Flight time: {flight_time:.2f} s")
print(f"Maximum altitude: {max_height:.1f} m")

self.plot_trajectory(trajectory, elevation, azimuth)
return elevation, azimuth, error

def plot_trajectory(self, trajectory, elevation, azimuth):
try:
plt.figure(figsize=(15, 5))

# 1.
plt.subplot(1, 3, 1)
plt.plot(trajectory[:, 0], trajectory[:, 1], 'b-', linewidth=2)
plt.plot([0, self.target_distance], [0, 0], 'g--', linewidth=2, label='Target direction')
plt.plot(self.target_distance, 0, 'ro', markersize=8, label='Target point
```

```
    ')
plt.plot(0, 0, 'go', markersize=8, label='Launch point')
plt.xlabel('East distance (m)')
plt.ylabel('North distance (m)')
plt.title('Trajectory Top View')
plt.grid(True)
plt.axis('equal')
plt.legend()

# 2.
plt.subplot(1, 3, 2)
plt.plot(trajectory[:, 0], trajectory[:, 2], 'b-', linewidth=2)
plt.plot([0, self.target_distance], [self.cannon_altitude, self.
    target_altitude], 'g--', linewidth=2,
label='Line of sight')
plt.plot(self.target_distance, self.target_altitude, 'ro', markersize=8,
    label='Target point')
plt.plot(0, self.cannon_altitude, 'go', markersize=8, label='Launch point
    ')
plt.xlabel('Horizontal distance (m)')
plt.ylabel('Absolute altitude (m)')
plt.title('Trajectory Side View (Absolute Altitude)')
plt.grid(True)
plt.legend()

# 3.
plt.subplot(1, 3, 3)
plt.plot(trajectory[:, 1], trajectory[:, 2], 'b-', linewidth=2)
plt.plot(0, self.target_altitude, 'ro', markersize=8, label='Target point
    ')
plt.xlabel('Lateral offset (m)')
plt.ylabel('Absolute altitude (m)')
plt.title('Trajectory Front View')
plt.grid(True)
plt.legend()

plt.tight_layout()
plt.show()
except Exception as e:
print(f"Plotting failed: {e}")


def main():
print("Artillery Ballistics Calculator - Wind and Altitude Effects")
print("=" * 60)

ballistics = WindBallistics()

print("Please enter the following parameters:")

wind_speed = float(input("Wind speed (m/s): "))
wind_direction = float(
input("Wind direction (degrees, 0=headwind, 90=crosswind from left, 180=
    tailwind, 270=crosswind from right): "))
```

```
target_distance = float(input("Target distance (m): "))
cannon_altitude = float(input("Cannon altitude (m): "))
target_altitude = float(input("Target altitude (m): "))
muzzle_velocity = float(input("Muzzle velocity (m/s): "))

ballistics.set_parameters(wind_speed, wind_direction, target_distance,
cannon_altitude, target_altitude, muzzle_velocity)

elevation, azimuth, error = ballistics.calculate_solution()

print("\nFiring instructions:")
print("=" * 50)
print(f"1. Set cannon elevation to: {elevation:.1f}°")
print(f"2. Set cannon azimuth to: {azimuth:.1f}° (relative to target
    direction)")
print(f"3. Fire with muzzle velocity: {muzzle_velocity} m/s")

if error < 10:
print("\nPrediction accuracy: High (error < 10m)")
elif error < 50:
print("\nPrediction accuracy: Medium (error < 50m)")
else:
print(
"\nPrediction accuracy: Low (error >= 50m), consider adjusting parameters
     or using higher muzzle velocity")


if __name__ == "__main__":
main()
```