Research on the Optimal Firing Angle and Initial Velocity of Vintage Artillery Based on Numerical Calculation

Team 796, Problem B

November 9, 2025

Abstract

We conceive a vintage artillery piece from roughly two centuries ago and, under multiple physical factors (aerodynamic drag, wind, projectile spin, and target elevation), investigate the optimal launch angle and optimal muzzle velocity. We employ the embedded Runge–Kutta–Fehlberg method (RK45) to numerically integrate the governing equations of motion and derive simplified, easy-to-use approximation formulas.

First, with the gun and target at the same elevation, no wind, and no spin, and with aerodynamic drag proportional to the square of speed, we formulate the ordinary differential equations and solve them numerically using Python's ODE routines. We plot the launch-angle–velocity relationships and introduce performance metrics such as flight time and terminal kinetic energy in order to identify an optimal launch angle. For a representative case, we find that at a muzzle velocity of approximately 450 m/s and a launch angle of 2.5–2.8°, performance is near-optimal.

Next, we incorporate wind by replacing the speed in the drag term with the relative air–projectile speed. We find that for shallow angles (<10°) the wind has only a minor influence on the trajectory (percent differences under 3%), whereas for steeper angles (>40°) the discrepancy grows significantly.
We then include projectile spin and the associated Magnus force. The influence of spin likewise grows with launch angle.

Fixing the muzzle velocity at 450 m/s, we fit the range–angle curve and obtain a compact regression model，reaching the function $\theta_0 \approx \frac{x}{282} + 1.016$。

Building on this, we consider nonzero elevation differences between gun and target at a fixed horizontal distance x = 1200 m and directly generate the required angle–velocity combinations.：

$$v_0(\theta_0,h)=0.0010\theta_0{}^3+-0.1026\,\theta_0{}^2+C4.25\theta_0+(0.0042h^2-0.315h+428.7)$$

$$\theta_0\approx\frac{h+161.02}{38.01}=\frac{h}{38.01}+4.236$$

Then,we We establish a more sophisticated and comprehensive model at a fixed v_0=450m/s. Finally, we introduce correction terms for changing target range and wind, thereby providing artillery officers a convenient field-calculation rule of thumb that requires little calculator：

$$\beta=\frac{|\vec{u}|}{5}\frac{0.0006\theta_0{}^3-0.072\theta_0{}^2+2.85\theta_0+2.15}{x}$$

Keywords: projectile trajectory; aerodynamic drag; numerical integration; optimal launch angle; optimal muzzle velocity.

1 Introduction

1.1 Problem Restatement

Consider a 150–200-year-old artillery piece (Figure 1) that fires a spherical projectile of diameter 11 cm and mass 5 kg. Determine the launch angle and muzzle velocity under the following scenarios:

① Gun and target are both at an elevation of 500 m, separated by a horizontal distance of 1200 m.

② The horizontal distance lies between 1000 and 1500 m; the gun and target are at different elevations, and wind is present.

Additionally, provide a simplified procedure that enables an officer with basic mathematical training to estimate the required launch angle and muzzle velocity rapidly without a calculator.

figure 1 Dagu Fort（1841）[1]

1.2 Background Research

Determining the projectile's point of impact is a prototypical problem in exterior ballistics. Although our focus is not directly on vintage artillery, the analytical and computational techniques developed in this mature field are clearly instructive. This section therefore undertakes a broad literature review and background survey to lay the groundwork for the modeling that follows.

1.2.1 Quadratic-drag model

Projectile impact prediction is a classical problem of exterior ballistics. Rather than ideal free-flight motion, the projectile experiences aerodynamic drag; a commonly used model assumes a drag magnitude proportional to the square of the speed. This model describes mid-to-high-speed flight (particularly for large-caliber shells) effectively. Prior work shows that for low-elevation trajectories where the horizontal component dominates, one can derive approximate closed forms using the Lambert W function[2]. Chudinov and others systematically studied projectile motion in a quadratic-drag medium, accounting for Mach-number dependence of the drag coefficient and shape parameters, and constructed approximations for flight time, apogee, range, and the optimal launch angle across a wide range of initial conditions[3].

1.2.2 Vintage artillery and spin

By the mid-19th century, rifled barrels had matured and been widely deployed[4]; our study period coincides with these developments. Rifling imparts spin to the projectile, giving rise to the Magnus force as well as spin-damping and rolling moments, all of which can perturb the trajectory. Crosswinds, veering winds, and head/tail winds also introduce characteristic disturbances.

As for sighting in early artillery, aiming devices were rudimentary (e.g., simple front and rear sights or improvised fixtures aligned with stars and the barrel centerline). Muzzle velocity was roughly adjustable by varying the propellant charge[5]; although imprecise, we may reasonably assume a continuously tunable range up to about 450 m/s.

1.2.3 Historical measurement capability

(1) Maps with known scales allow reasonably accurate measurement of horizontal separation.
(2) Topographic maps provide site elevations, hence the elevation difference between gun and target.
(3) Wind speed and direction can be estimated from flags, smoke plumes, vegetation, or dust; a simple handheld anemometer can also be employed. It is worth noting that in reality, people in ancient times only measured wind directions parallel to the ground (such as southeasterly winds).

1.2.4 Advances in numerical simulation

With modern numerical methods and CFD, research has increasingly combined high-fidelity simulation with experiment—for example, calibrating CFD models for small-caliber bullets, measuring muzzle-velocity decay, and correlating drag coefficients with bullet shape[6]. Comparative studies indicate that embedded RK4(5) solvers (RK45) achieve good accuracy for artillery trajectories when paired with appropriate error control[7].

In summary, we identify at least the following factors that must be considered:

① aerodynamic drag proportional to the square of speed;

② the influence of wind speed and direction on the trajectory;

③ the influence of projectile spin on the trajectory;

④ air-density variation induced by altitude;

⑤ the relationship between Mach number and the drag coefficient;

⑥ the limited adjustment precision of historical artillery platforms, which must be reflected in practical recommendations.

Introducing the above parameters yields a system of ordinary differential equations, which we solve numerically using Python ODE solvers in conjunction with the RK45 method. On the basis of these numerical solutions, we then investigate the governing regularities.

1.3 Problem Analysis

The forces acting on the projectile are sufficiently complex that a stepwise modeling strategy is advisable: start from a baseline model and then introduce realistic effects incrementally, quantifying each contribution. Finally, distill the results into compact approximations usable in the field.

Layer 1: Set the elevation difference to zero and consider only gravity and quadratic drag. For each fixed range, one obtains a family of (angle, velocity) pairs capable of reaching the target. We evaluate these pairs using three criteria—shortest flight time, largest terminal kinetic energy, and minimal peak altitude—and select an optimal operating point. The baseline model is deliberately simple and parameter-sparse, facilitating both integration and analysis.

Layer 2: Include crosswind, which adds a lateral acceleration and turns the 2D (x–y) problem into 3D (x–y–z); a small azimuthal correction to the aim is then required.

Layer 3: Include projectile spin and the Magnus force, which influences both vertical motion and lateral drift.

Layer 4: Introduce elevation differences and repeat the above analyses to obtain new (angle, velocity) curves as functions of the height offset h. These are assembled into lookup tables for rapid use.Also,we will provide the fitted approximate solution to facilitate rapid calculation.

At each layer of the model, we present the corresponding equations of motion and numerical solution procedure; in parallel, we conduct layer-wise comparisons to quantify the contribution of each force to the range and impact-point error, thereby determining the level of simplification acceptable for operational use or field estimation and the correction terms that remain necessary.

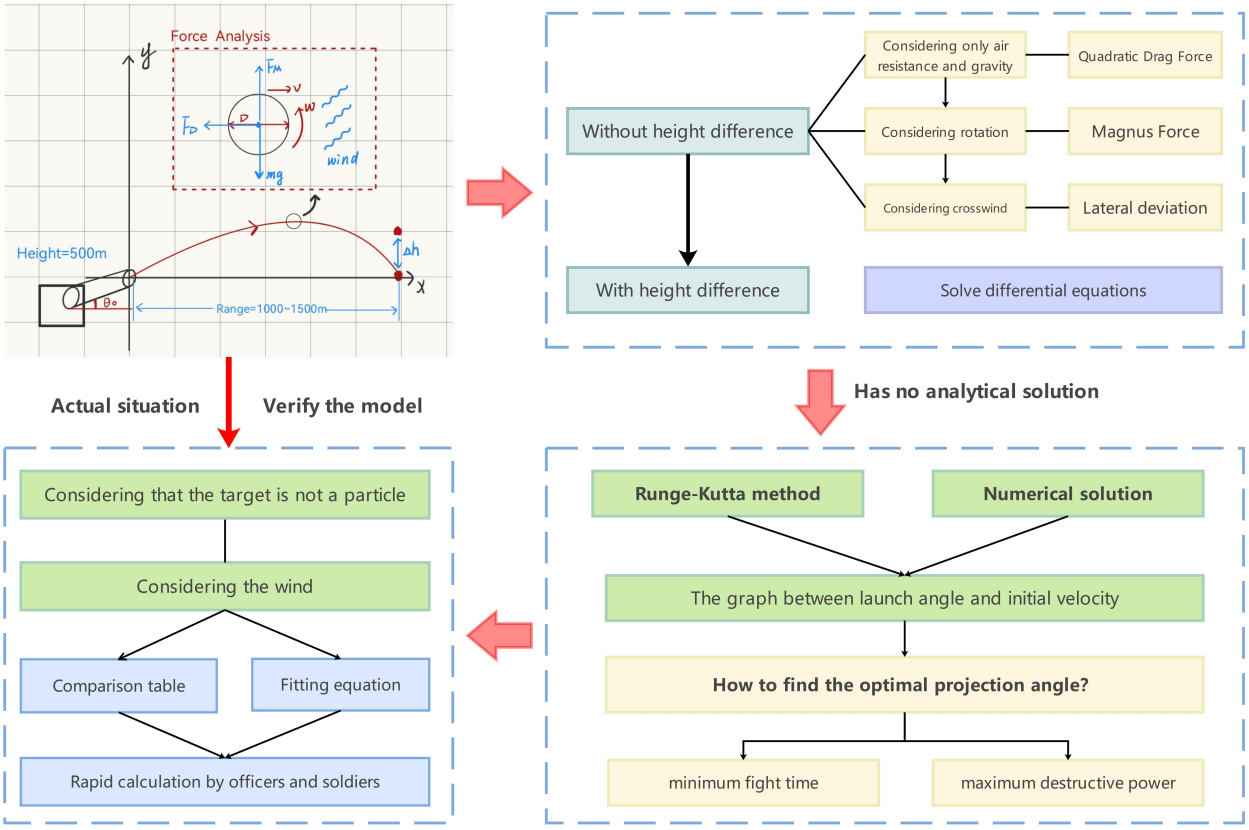


figure 2 Analysis of Problem

2 Assumption and Notation

2.1 Assumptions

① Elevation, wind speed, and horizontal distance are measurable.

② Temperature and humidity are constant.

③ No obstacles are present along the line of fire.

④ Gravitational acceleration is 9.81 m/s².

⑤ Muzzle velocity is continuously tunable.

⑥In ancient times, when measuring wind speed, only the wind coming from the horizontal direction was reported, and vertical wind was not taken into account.

2.2 Notations

table 1 Notations

Symbol	Definition	Numeric Value
$\vec{r}$	Position vector of the projectile	
$\vec{v}$	Velocity vector of the projectile	
A	Cross-sectional area of the projectile	
ρ	Air density	Depends on height
C_d	Drag coefficient	Depends on Mach number
$\overrightarrow{F_D}$	Drag force vector	Opposes velocity direction
$\vec{g}$	Gravitational acceleration vector	9.81m/s^2
$\vec{F}$	Drag force vector(considering wind)	
$\overrightarrow{F_M}$	Magnus force vector	
$\vec{v}_{rel}$	Relative velocity vector	
θ_0	Initial launch angle	（0,90°）
$\overrightarrow{v_0}$	Initial velocity vector of the projectile	Up to 450m/s

3 Case I: Gun and Target at the Same Elevation

3.1 No Wind

Given a projectile of diameter D=11cm，and mass m=5kg，with air density at an elevation of 500m of $\rho = 1.175 kg/m^3$，let v denote the instantaneous speed and $A = \pi D^2/4A$ the frontal area

According to the literature ,the drag coefficient depends on the Mach.

Number Ma（normalized by the local speed of sound）:

$$Ma = \frac{v}{v_{声}} \quad \text{(equation 1)}$$

Satisfies the following relation：

$$C_d = \begin{cases} 0.43 & Ma < 0.3 \\ 0.516 - 0.625Ma + 1.085Ma^2 & 0.3 \leqslant Ma < 1.1 \\ 1.247 - 0.101Ma + 0.008Ma^2 & 1.1 \leqslant Ma \leqslant 6.0 \end{cases} \quad \text{(equation 2)}$$

Vector Form of the Air Resistance Force：

$$\overrightarrow{F_D} = -\frac{1}{2}\rho C_d A v\vec{v} \quad \text{(equation 3)}$$

Kinematic Equations：

$$m\frac{d\vec{v}}{dt} = -\frac{1}{2}\rho C_d A v\vec{v} + m\vec{g} \quad \text{(equation 4)}$$

Definition of the Damping Coefficient：

$$\beta = \frac{\rho C_d \pi D^2}{8m} \quad \text{(equation 5)}$$

Simplification of the Equation of Motion：

$$\frac{d\vec{v}}{dt} = -\beta v\vec{v} + \vec{g} \quad \text{(equation 6)}$$

Magnitude of Velocity：

$$v \equiv |\vec{v}| = \sqrt{v_x^2 + v_y^2} \quad \text{(equation 7)}$$

Relationship Between Displacement and Velocity：

$$\frac{d\vec{r}}{dt} = \vec{v} \quad \text{(equation 8)}$$

System of Differential Equations Obtained：

$$\begin{cases} \frac{dx}{dt}=v_x, v_{x0} = v_0\cos\theta_0 \\ \frac{dy}{dt}=v_y, v_{y0} = v_0\sin\theta_0 \\ \frac{dv_x}{dt}=-\beta\sqrt{v_x^2+v_y^2}\, v_x, \\ \frac{dv_y}{dt}=-g-\beta\sqrt{v_x^2+v_y^2}\, v_y. \end{cases} \quad \text{(equation 9)}$$

Set $x = 1200, y = 0$ to solve the initial value problem: v_0 and θ_0 are our target parameters .

This equation is transcendental and lacks an analytical solution; therefore, we employ numerical computation methods, utilizing the RK45 algorithm and Python's ODE solver. The specific procedure is as follows:

Based on the aforementioned system of first-order ordinary differential equations, we use embedded fourth- and fifth-order solutions to compare and estimate errors, adaptively adjust the step size, and combine preset accuracy tolerances (rtol=1e–5, atol=1e–7) with a maximum step size (0.05 s) to balance accuracy and computational speed. The global truncation error reaches $O(h^4)$(where hh is the set step size). By identifying the intersection point of the vertical displacement and the target height, combined with linear interpolation, the precise range is extracted.

Regarding the error of the RK45 algorithm, with h = 0.05s, the numerical truncation error obtained through step-size convergence testing is $\leq 0{,}7$m, which is already a relatively accurate value.

The following is referred to as the perceptual process, aimed at intuitively sensing the values of physical quantities and establishing a physical understanding.

Set the initial velocity $v_0 = 300$m/s , and compare the motion trajectories with air resistance (solid line) and without air resistance (dashed line) at launch angles of 30°, 45°, and 60°. The curves shown in Figure 3 are obtained.

The fundamental insights that can be established are as follows:

① When air resistance is present, the range of the projectile is significantly reduced, from nearly 10,000 meters to approximately 3,000 meters.②In the absence of air resistance, it is well known that the range x is given by$x = \frac{v_0^2 \sin 2\theta}{g}$, Evidently, the maximum range is achieved at θ=45°.

However, when air resistance is considered, the maximum range does not necessarily occur at45° , a point also noted in multiple references [8, 9]

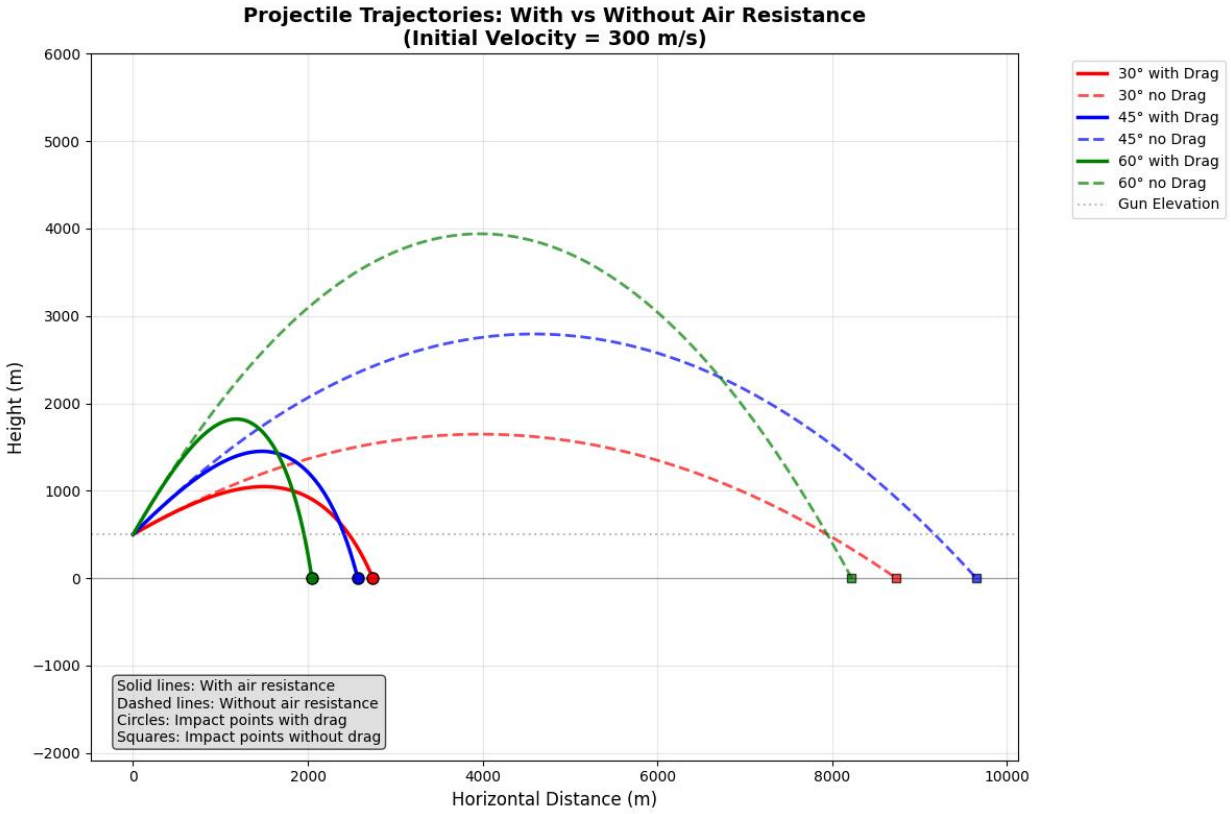


figure 3 Analysis of Trajectories with and without Drag

Next, we investigate the relationship between maximum range, launch angle, and initial velocity to further establish fundamental understanding. Adhering to the principle of controlling variables, we keep the velocity constant while varying the launch angle to obtain the curve in Figure 4(a), and keep the launch angle constant while varying the initial velocity to obtain the curve in Figure 3(b).

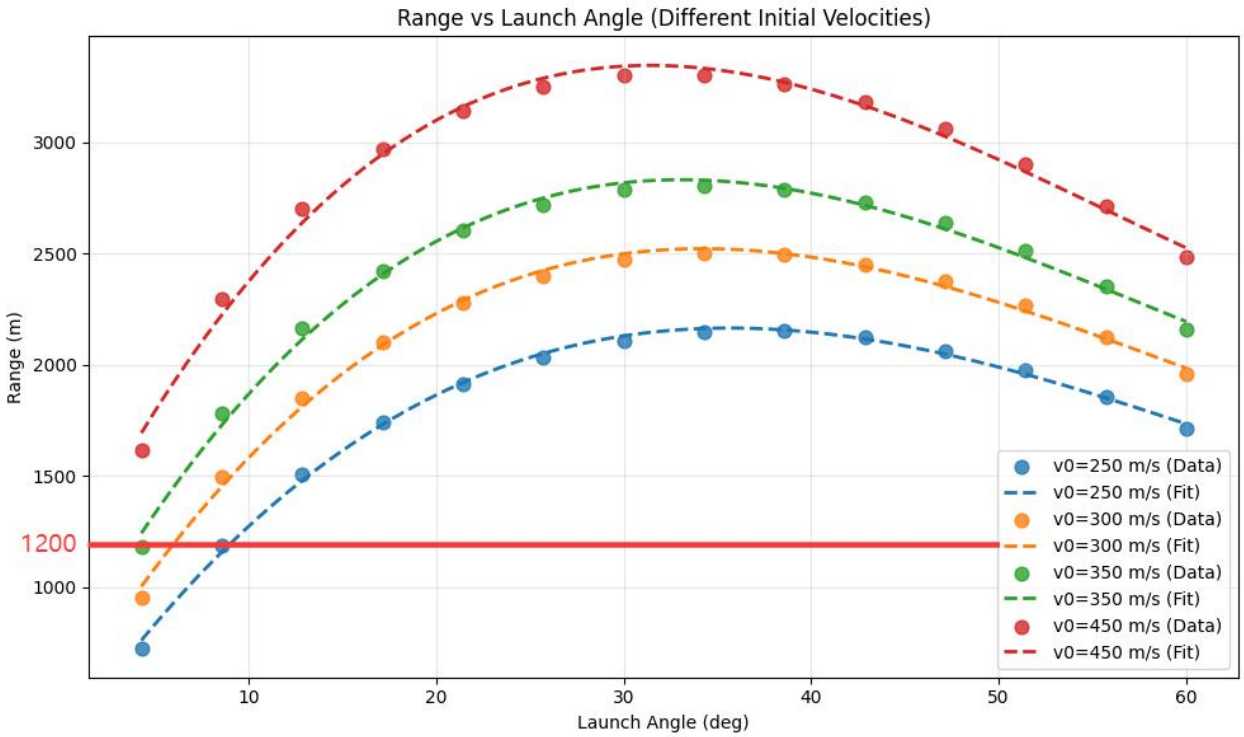
The following conclusions can be drawn from the figures:

①The greater the initial velocity, the farther the range

②As the launch angle increases, the range first increases and then decreases, exhibiting an extremum point (less than 45°), and the position of this extremum depends on the magnitude of the initial velocity.

Based on the above observations, we can consider that for the two graphs in Figure 4, drawing a vertical line at 1200 m parallel to the y-axis will intersect a family of curves, corresponding to a series of (θ_0, v_0)combinations， This allows us to plot a $\theta_0 - v_0$curve， On this curve, the projectile can always reach the target location. However, we need to select an optimal$\theta_0$and v_0 based on certain evaluation metrics.

(a)With the initial velocity kept constant, the launch angle is varied.



(b)With the launch angle kept constant, the initial velocity is varied.

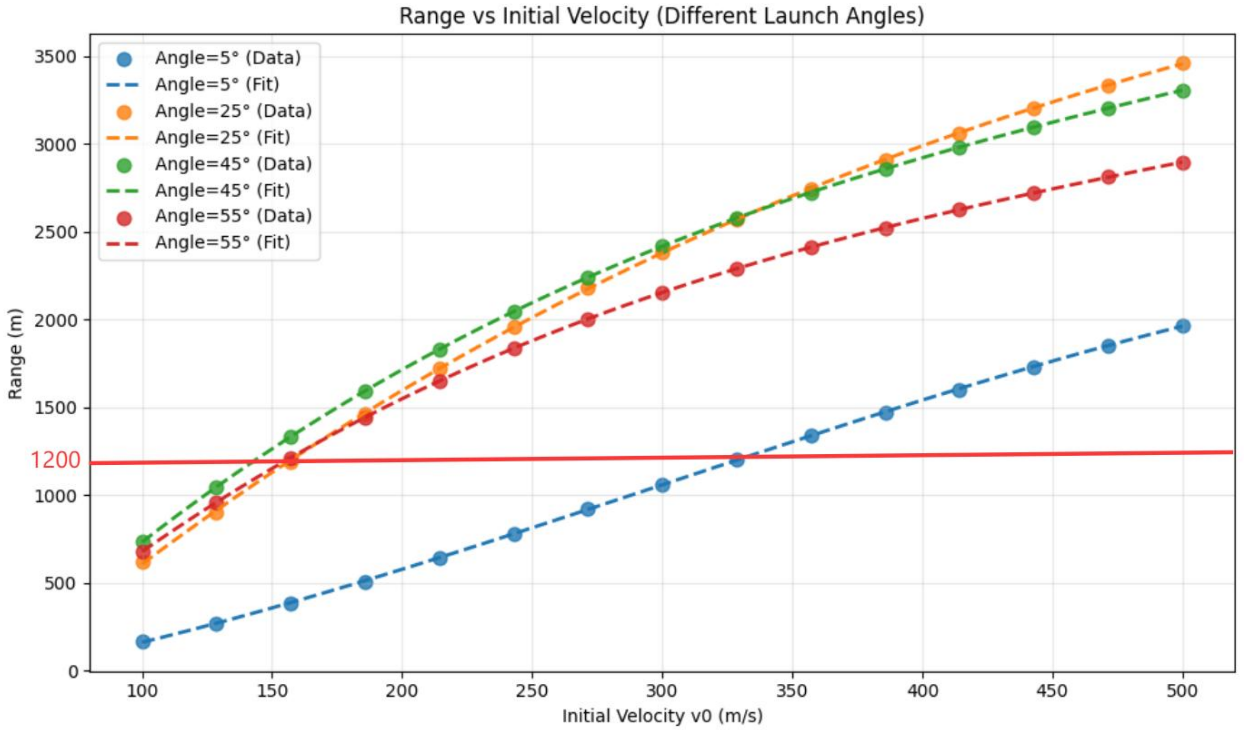


figure 4 Maximum Range Under Different Conditions

Similarly, the RK45 algorithm is employed to solve this initial value problem. Initially, without considering the variation in air resistance due to altitude changes (though the maximum height reached by the projectile at different angles is computed), the results shown in Figure 5 are obtained. The following conclusions can be drawn:

①The curve exhibits a U-shape, with the launch angle reaching an extremum (approximately 40°) at which the required initial velocity is minimized (around 100 m/s).
②The smaller the launch angle θ_0, the lower the maximum altitude the projectile can reach (as low as approximately 17 m). When $\theta_0$approaches 80 m/s, the maximum altitude can reach up to 2 km. This lays the foundation for subsequent models that incorporate altitude differences.

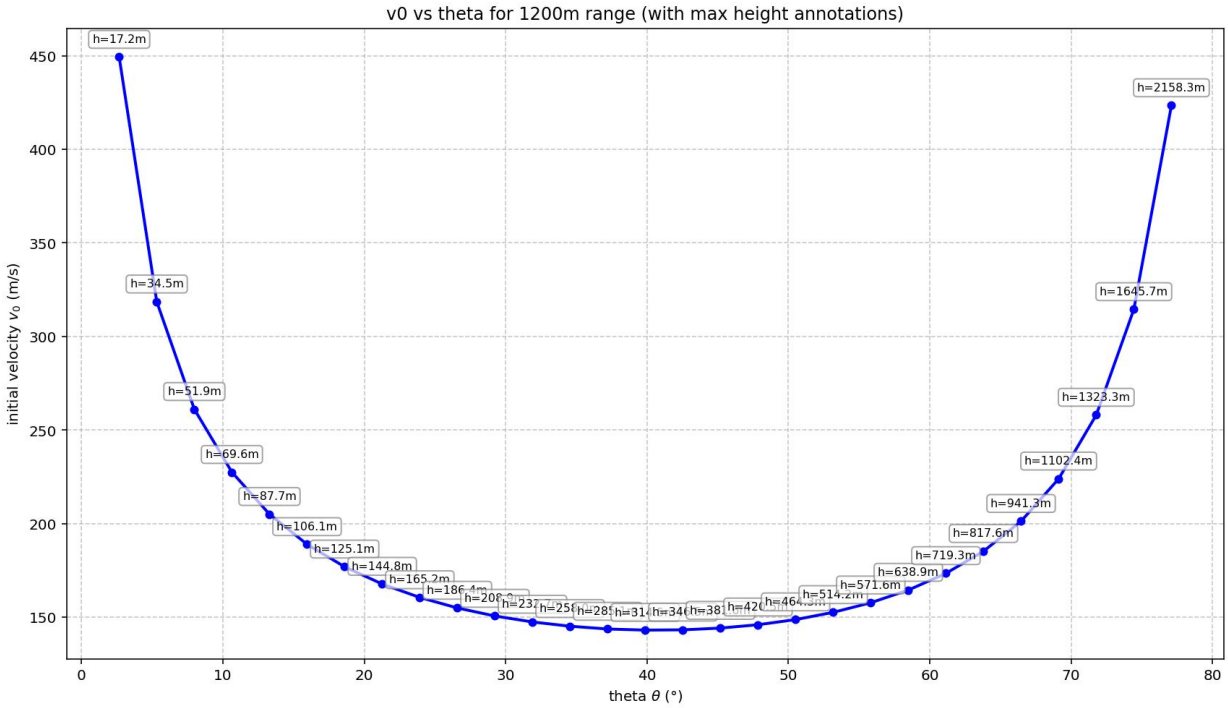


figure 5 Launch Angle-Initial Velocity Curve at 500 Meters Altitude

Understanding this phenomenon is relatively straightforward: when v_0 is sufficiently large—comparable to the muzzle velocity of a handgun, which can reach 700 m/s [10], similar in magnitude to the artillery's 450 m/s—the flight time is very short, and the vertical drop is minimal. In this case, near-horizontal firing can effectively hit the target, requiring only a very small angle (less than 5°) to counteract the drop due to gravity. Conversely, when $\theta_0$is large, the horizontal velocity component is small, while the vertical component is significant. This results in a longer flight time to reach the target, necessitating a larger initial velocity.

Considering that altitude affects air density, which in turn influences air resistance, we examine the $\theta_0 - v_0$ curves under different air density conditions. Based on previous calculations, the projectile can reach a maximum altitude of 2 km. Referring to relevant data, the variation of air density with altitude is obtained as follows:

table 2 Air Density at Different Altitudes

Altitudes（m）	Air Density （kg/m^3）
500	1.175
1000	1.12
1500	1.065
2000	1.013
2500	0.937

Selecting the same angle, the percentage deviation of the initial velocity is defined as:

$$\text{Percentage Difference} = \frac{v_i - v_0}{v_0} \quad \text{(equation 10)}$$

where v_0 is the initial velocity at an altitude of 500 m, and v_i is the initial velocity at other altitudes. Through this percentage deviation, we can intuitively observe the impact of altitude variation on the initial velocity.

The resulting curve, shown in Figure 6, reveals the following:

① For altitude variations within 500 m and launch angles θ_0 less than 50°, the percentage deviation in initial velocity is approximately 2%. Under practical conditions, with an initial velocity of 300 m/s, this results in an absolute error of 6 m/s. However, given the limited precision of historical artillery, this velocity deviation is negligible.

② When the altitude variation exceeds 500 m, the percentage deviation in initial velocity can reach 5%, corresponding to an absolute error of 15 m/s. This indicates that altitude variation has a significant impact on the projectile trajectory, necessitating consideration of variable air density.

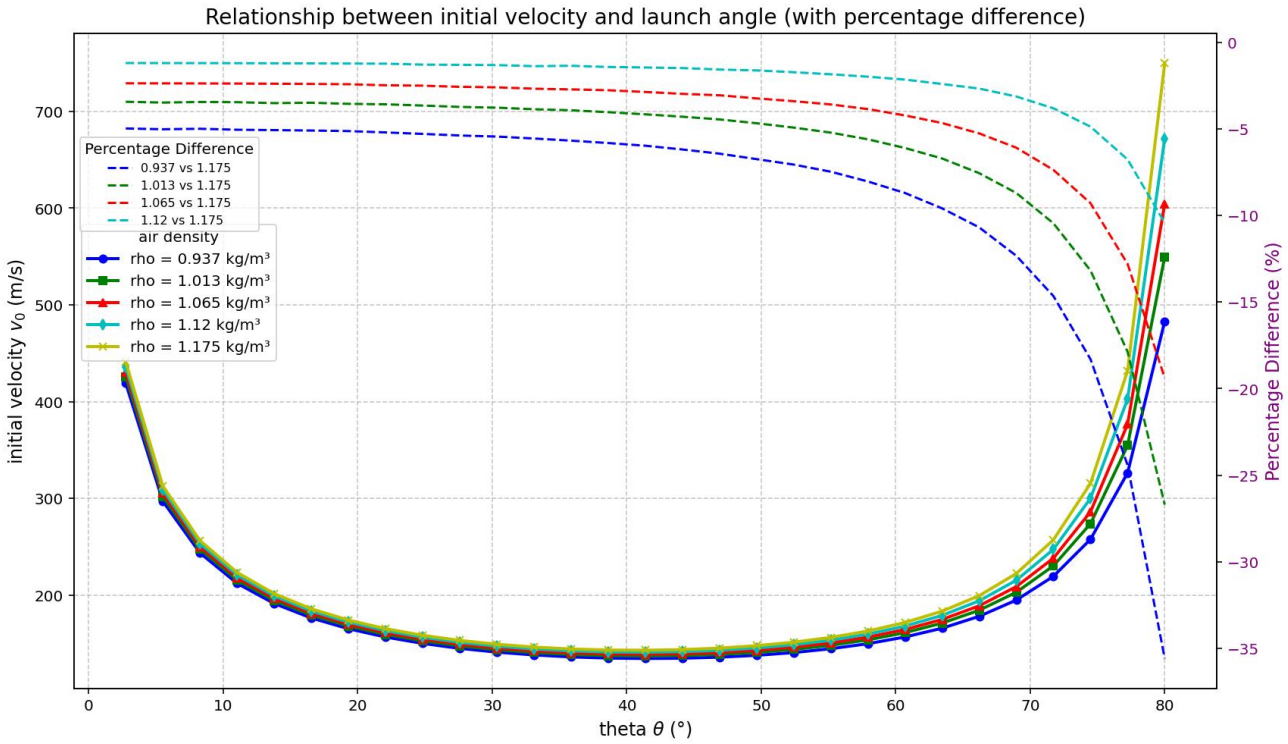


figure 6 Launch Angle-Initial Velocity Relationship at Different Altitudes

Similarly, we examine the influence of the drag coefficient on the projectile trajectory, as shown in Figure 7. It is observed that, under the same conditions, when the launch angle is less than 40° and the drag coefficient varies between 0.47 and 0.5, its effect on the initial launch velocity is not significant.

For the $\theta_0 - v_0$ curve, determining the optimal launch angle requires consideration of the following factors:

① The flight time of the projectile. On the battlefield, speed is critical; the faster the target can be engaged, the more effective the outcome.

② The lethality upon impact. Using the RK45 algorithm to calculate the velocity at impact, the kinetic energy of the projectile is derived. Greater kinetic energy corresponds to higher lethality.

③ The ascent height of the projectile. Higher ascent leads to greater variations in air resistance, making the actual trajectory more difficult to predict.

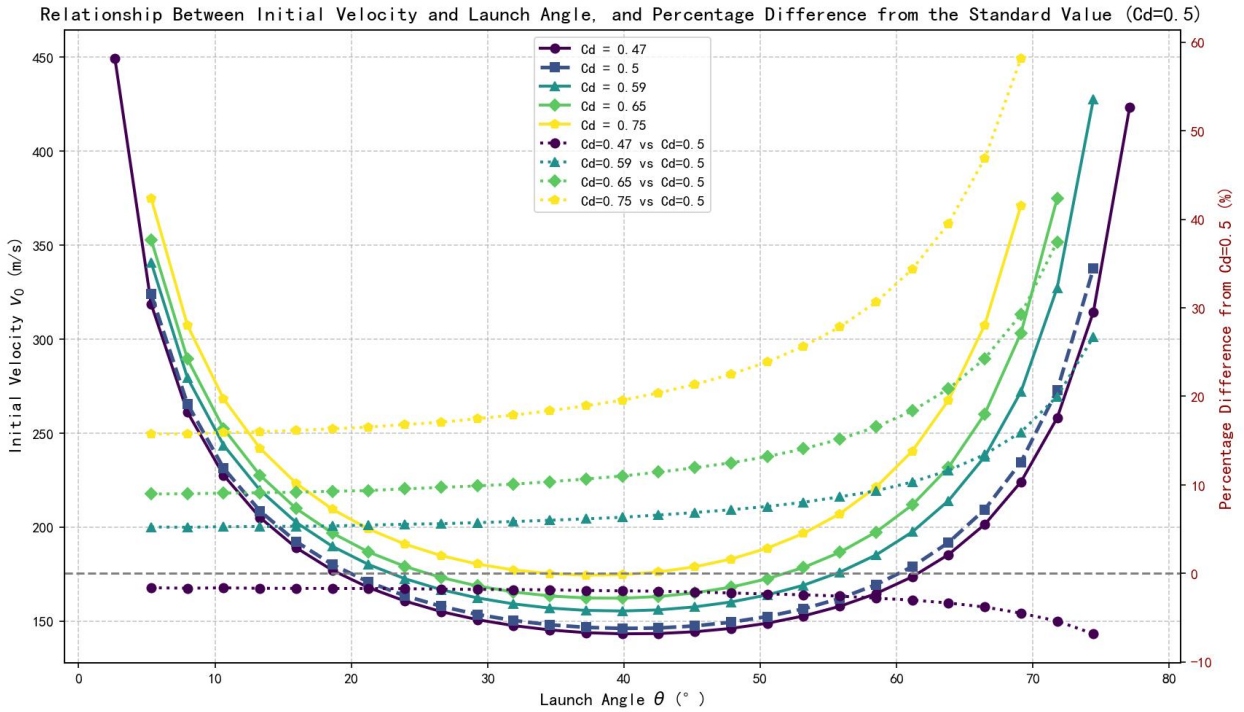


figure 7 Launch Angle-Initial Velocity Relationship at Cd

Integrating the parameters of flight time and kinetic energy at impact into the $\theta_0 - v_0(x)$ curve, the analysis reveals that an initial velocity of 450 m/s and a launch angle of 2.66° achieve the maximum lethality (i.e., kinetic energy), the shortest flight time, and the lowest ascent height, resulting in the best overall performance.

Solving the model for $x = 1200$ m, the optimal launch angle and initial velocity are determined to be $\theta_0 = 2.66°$ and $v_0 = 450$ m/s. This outcome can be analogized to pistol shooting, where direct aiming at the target achieves effective engagement.

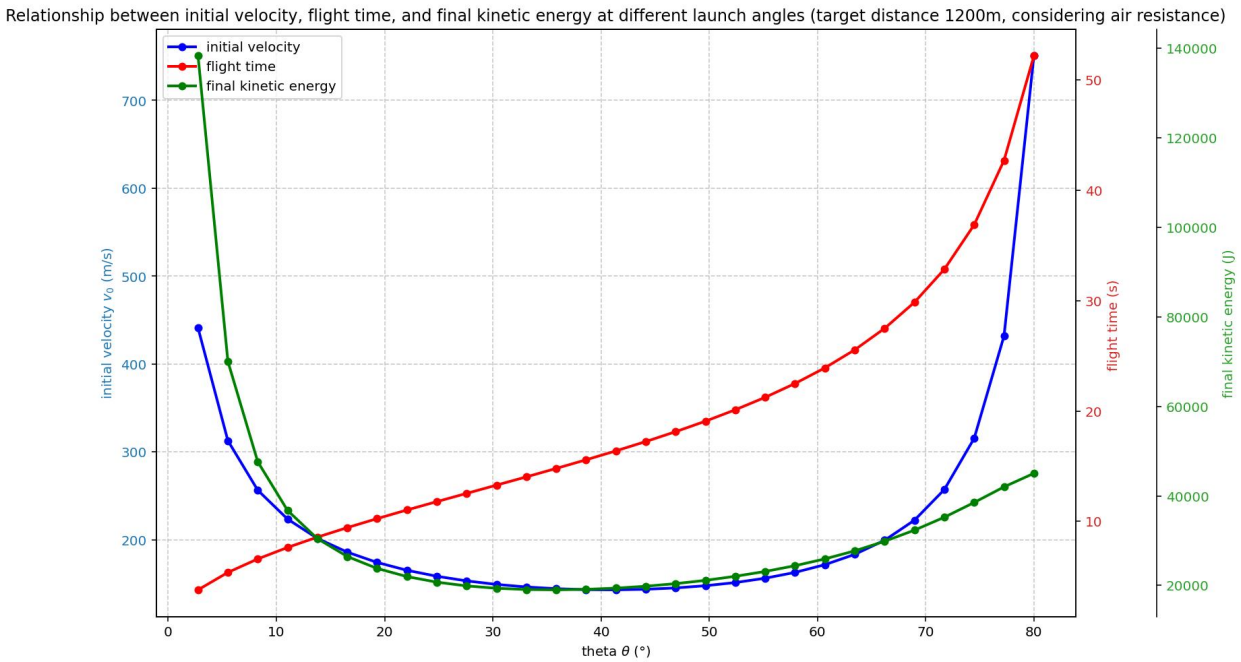


figure 8 The curves of flight time and landing kinetic energy varying with the launch parameters

3.2 Considering the Effect of Wind

After introducing wind, the model becomes more complex, necessitating the use of a three -dimensional coordinate system, as illustrated in Figure 9.

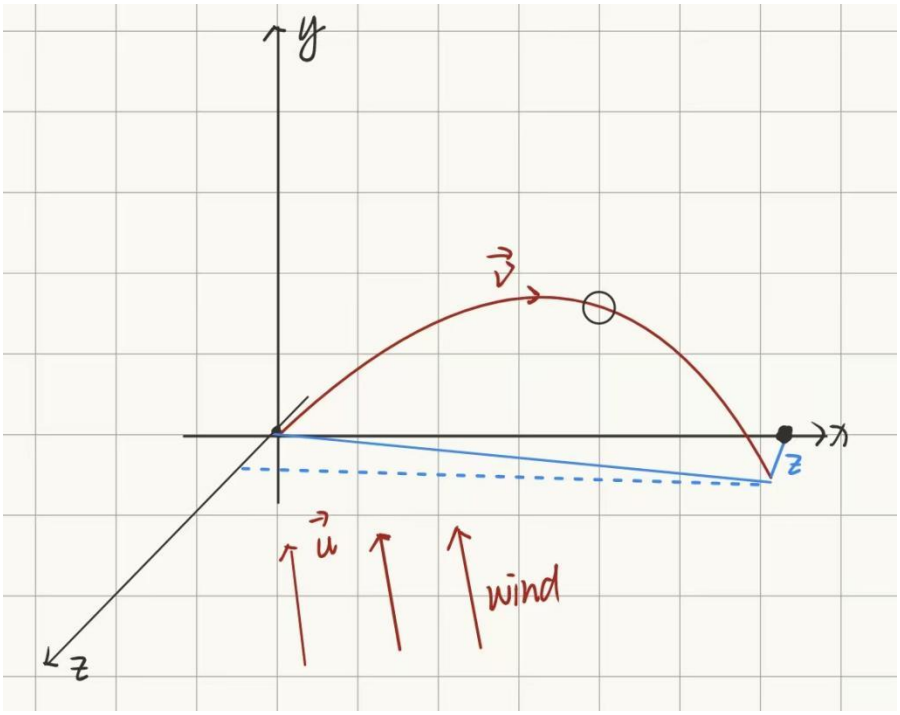


figure 9 Three-dimensional coordinate system

The essence of wind force lies in air resistance. Therefore, incorporating wind solely requires modifying the air resistance term.

Let $\vec{F}$, denote the air resistance after introducing wind. The velocity should be replaced by the relative velocity $\vec{v}_{rel} = \vec{v}_0 - \vec{u}$, where $\vec{u}$ represents the wind velocity vector.

$$\vec{F} = -\frac{1}{2} C_d \rho A v_{rel} \cdot \vec{v}_{rel} \quad \text{(equation 11)}$$

$$m\frac{d\vec{v}}{dt} = -mg\vec{j} - \frac{1}{2} C_d \rho A v_{rel} \cdot (\vec{v} - \vec{u}) \quad \text{(equation 12)}$$

$$v_{rel} = \sqrt{(v_x - u_x)^2 + (v_y - u_y)^2 + (v_z - u_z)^2} \quad \text{(equation 13)}$$

Final System of Differential Equations:

$$\begin{cases} \frac{dx}{dt} = v_x \\ \frac{dy}{dt} = v_y \\ \frac{dz}{dt} = v_z \\ m\frac{dv_x}{dt} = -\frac{1}{2} C_d \rho A \cdot (v_x - u_x) \cdot v_{rel} \\ m\frac{dv_y}{dt} = -mg - \frac{1}{2} C_d \rho A \cdot (v_y - u_y) \cdot v_{rel} \\ m\frac{dv_z}{dt} = -\frac{1}{2} C_d \rho A \cdot (v_z - u_z) \cdot v_{rel} \end{cases} \quad \text{(equation 14)}$$

According to literature, typical wind speeds range from 5 to 7 m/s (excluding severe weather conditions).

An offset along the z-axis is observed, which implies that the horizontal launch angle alone is insufficient to fully characterize the launch orientation. A correction angle must be introduced. Let β be defined as the angle between the initial velocity vector of the projectile and the yoz-plane. When the target is reached, the offset along the z-axis is z. Given that z is much smaller than the range x, β is small, and the trajectory can be considered approximately parallel to the xoy-plane. Thus:

$$\beta \approx \frac{z}{x} \quad \text{(equation 15)}$$

With the introduction of a 5 m/s wind speed along the z-axis, the resulting offset in the z-axis is as small as 2.1 m, corresponding to $\beta = 0.0168$. If the target object is sufficiently large and considering the limited adjustment precision of historical artillery, this 2.1 m offset may be considered negligible. However, as the launch angle increases, the z-axis offset grows significantly, making the correction for β essential.

Using an initial velocity of 450 m/s and a launch angle of 2.6196°, the difference compared to the scenario without wind is 0.04°

.

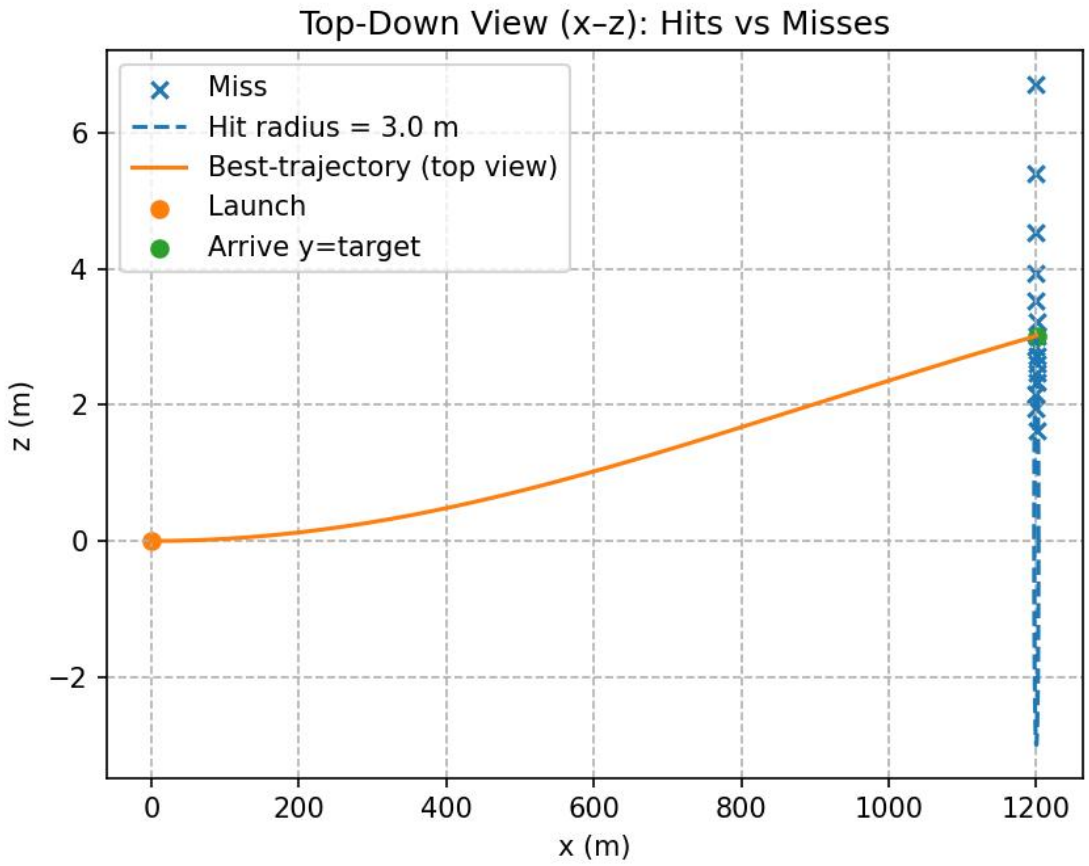


figure 10 Offset Caused by Wind in the Z-axis Direction

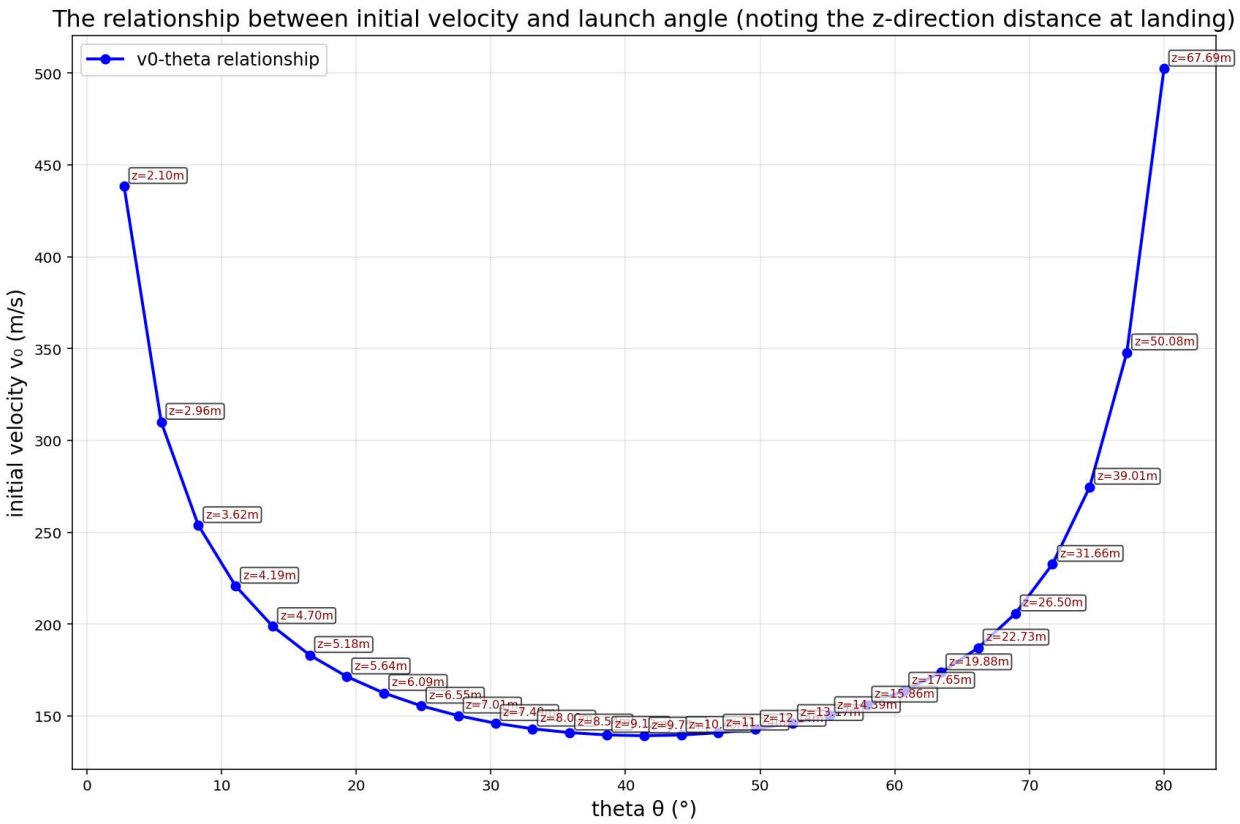


figure 11 Diagram of Launch Angle vs. Initial Velocity Considering Wind Speed

However, when θ_0 is relatively large, the offset in the z-direction becomes non-negligible. We obtain a fitted relationship as follows:

$$z(\theta_0) = 0.0006{\theta_0}^3 - 0.072{\theta_0}^2 + 2.85\theta_0 + 2.15 \qquad \text{(equation 16)}$$

Thus, the formula for the offset angle is derived as:

$$\beta \approx \frac{0.0006{\theta_0}^3 - 0.072{\theta_0}^2 + 2.85\theta_0 + 2.15}{x} \qquad \text{(equation 17)}$$

Assuming the offset along the z-axis due to wind is proportional to the wind speed, and based on the formula fitted for a wind speed of 5 m/s, the final expression is obtained as:

$$\beta = \frac{|w|}{5}\frac{0.0006{\theta_0}^3 - 0.072{\theta_0}^2 + 2.85\theta_0 + 2.15}{x} \qquad \text{(equation 18)}$$

3.3 Considering Spin

If the artillery is rifled, the projectile will spin along its axis of motion upon firing. In the presence of wind, this spin generates a Magnus force, which subsequently influences the equations of motion. The derivation proceeds as follows:

$$F_M = \frac{1}{2}\rho C_L(S, Re, Ma) A V_{rel}^2 \hat{n} \qquad \text{(equation 19)}$$

Where $\hat{n} = \frac{\vec{\omega}\times\vec{V}_{rel}}{|\vec{\omega}\times\vec{V}_{rel}|}$represents the direction of the Magnus force, and$C_L$is the lift coefficient, which depends on the shape parameter$S$,the Reynolds number Re,and the Mach number Ma , characterizing the aerodynamic properties. Here, $S = \frac{\alpha|\vec{\omega}|}{V}$, $Re = \frac{\rho VD}{\mu}$,

Moment of Inertia of a Sphere：

$$I = \frac{2}{5}mR^2 \qquad \text{(equation 20)}$$

Aerodynamic Torque:

$$T_W = -\frac{1}{2}\rho C_W A\omega V_{rel} \qquad \text{(equation 21)}$$

Where C_W is the drag coefficient， and the negative sign indicates that the torque acts opposite to the direction of the angular velocity w（thus resisting rotation）

$$C_W = 0.012\frac{Dw}{2v} \qquad \text{(equation 22)}$$

Equation of Rotational Motion:

$$I\dot{\omega} = -\frac{1}{2}\rho C_W A\omega V_{rel} \qquad \text{(equation 23)}$$

To simplify the equations, we define a time-varying coefficient:：

$$\lambda(t) = \frac{0.5\rho C_W A}{I}V(t) \qquad \text{(equation 24)}$$

$V(t) = |\vec{V}_{rel}(t)|$ be the time-varying magnitude of the relative velocity. The equation can be rewritten as:：

$$\dot{\omega} = -\lambda(t)\omega \qquad \text{(equation 25)}$$

Rewritten in integral form:

$$\omega(t) = \omega_0 \exp\left(-\int_0^t \lambda(\tau)d\tau\right) \qquad \text{(equation 26)}$$

If λ(t)is treated as a constant, then:：

$$\omega(t) = \omega_0 \exp(-\lambda t) \qquad \text{(equation 27)}$$

$$\begin{cases} m\dot{\vec{V}} = mg\hat{j} - \frac{1}{2}\rho C_D A V_{rel}\vec{V}_{rel} + \frac{1}{2m}\rho C_L A V_{rel}^2\hat{n}, \\ \dot{\vec{r}} = \vec{V}, \\ I\dot{\omega} = -\frac{1}{2}\rho C_W A\omega V_{rel} \end{cases} \qquad \text{(equation 28)}$$

The solution yields an initial velocity of 450 m/s and a launch angle of 2.7263°, which differs by 0.06° from the case without wind.

Finally, a comparison of three scenarios—considering only drag, incorporating wind without spin, and incorporating both wind and spin—reveals that, similarly, when the launch angle is less than 10°, the effects of wind and spin on the projectile trajectory are minimal. It is only at larger launch angles that wind and spin play a critical role.

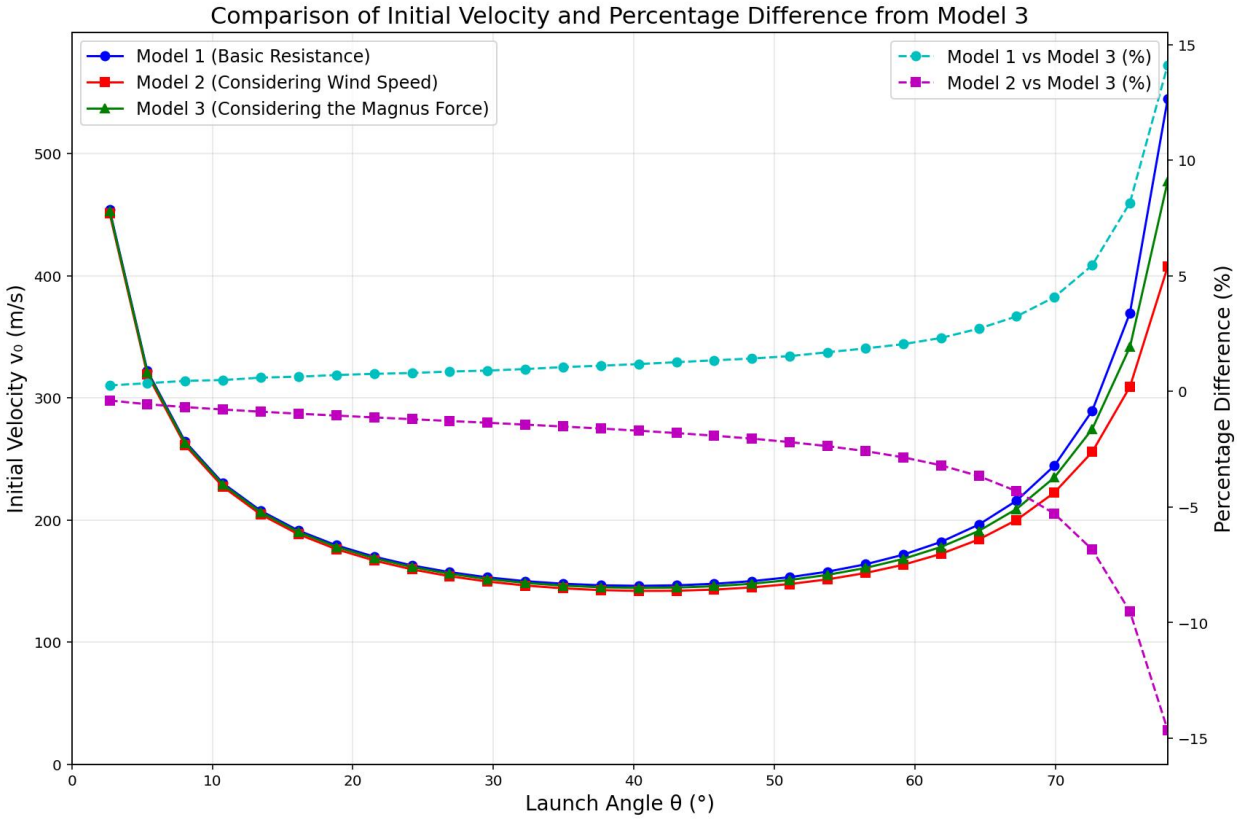


figure 12 Comparison of Initial Velocity-Launch Angle Curves Under Three Conditions

3.4 Fitting for Horizontal Distance

In the absence of altitude difference, based on previous conclusions, using an initial velocity of 450 m/s under near-horizontal firing conditions is optimal. Under such conditions, the flight time is short, the vertical height variation is minimal, and the influences of wind and projectile spin on the trajectory are negligible. Thus, the only variable is the horizontal distance x. For a given x, with$v_0$fixed at 450 m/s, the corresponding θ_0 needs to be determined. By plotting the $\theta_0 - x$ curve and fitting the data, the following relationship is obtained:

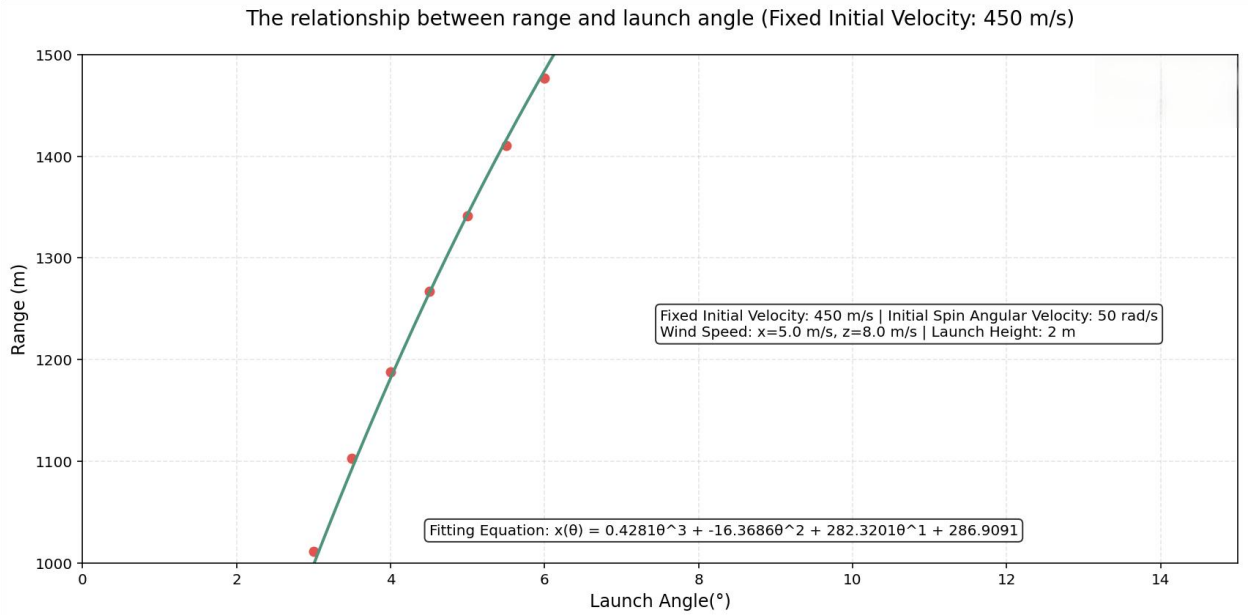


figure 13 Launch Angle-Range Curve

$$x(\theta_0) = 0.42818\,\theta_0^{\,3} + 16.36866\,\theta_0^{\,2} + 282.32010\,\theta_0 + 286.9091 \qquad \text{(equation 29)}$$

It can be observed that in the expression, the linear term dominates, with a percentage error of 4%. Considering only the linear term:

$$\theta_0 = \frac{x\text{-}286.9091}{282.32010} \approx \frac{x}{282} + 1.016 \qquad \text{(equation 30)}$$

4 Case with Altitude Differen

Without considering wind, the solution that accounts for height is obtained by changing the boundary condition of the differential equation from $y = 0$ to $y = h$. This yields the curve shown in Figure 13. For each specific height, a corresponding $\theta_0 - v_0$ curve can be extracted. A series of data points (see Appendix) are sampled from these curves. For each fixed height, a quadratic function is used for fitting, and the coefficient of determination R^2 is calculated. The results are presented in Table 3.

$$v_0(\theta_0) = A\,\theta_0^{\,3} + B\,\theta_0^{\,2} + C\,\theta_0 + D \qquad \text{(equation 31)}$$

table 3 Fitting Parameters at Different Heights

h（m）	A	B	C	D	R^2
0	0.0008	-0.083	3.42	405.6	0.9992
50	0.0012	-0.128	5.23	782.5	0.9991
-50	0.0006	-0.062	2.61	478.3	0.9993

100	0.0015	-0.157	6.42	557.2	0.9990
-100	0.0004	-0.047	2.03	337.6	0.9994
200	0.0020	-0.205	8.53	393.4	0.9989
-200	0.0003	-0.036	1.52	238.5	0.9994

It is observed that in Equation 28, the constant term D dominates. We now fit D against h, resulting in the following expression:

$$D(h) = 0.0042h^2 - 0.315h + 428.7 \quad \text{(equation 32)}$$

For parameters A, B, and C, which play secondary roles, average values are adopted. The final formulation is as follows:

$$v_0(\theta_0, h) = 0.0010\theta_0^{\,3} + -0.1026\,\theta_0^{\,2} + C4.25\theta_0 + (0.0042h^2 - 0.315h + 428.7)$$

(equation 33)

Substituting the data into the calculation yields an average percentage error of 2.23%, indicating a good fitting effect.

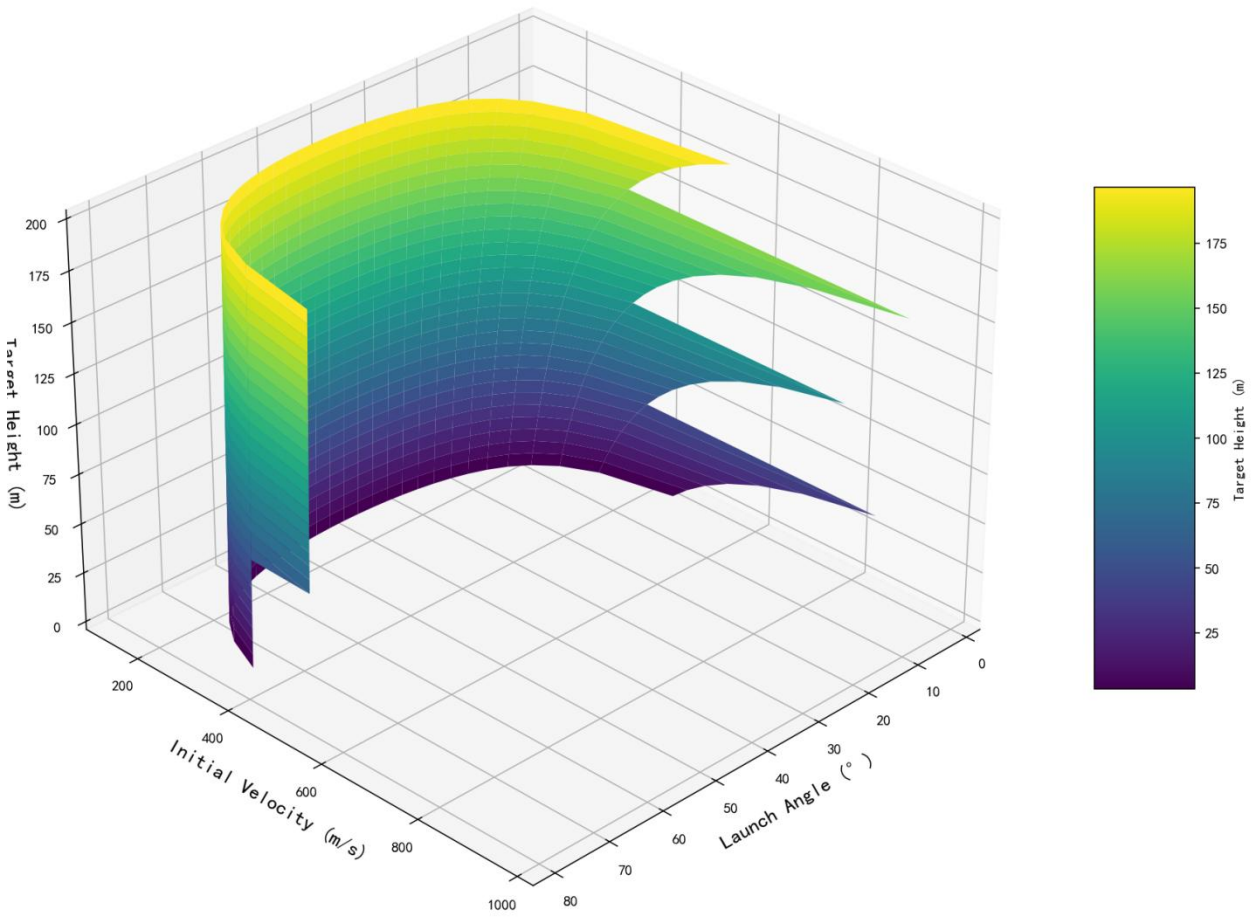


figure 14 Initial Velocity vs. Launch Angle Curve Considering Altitude

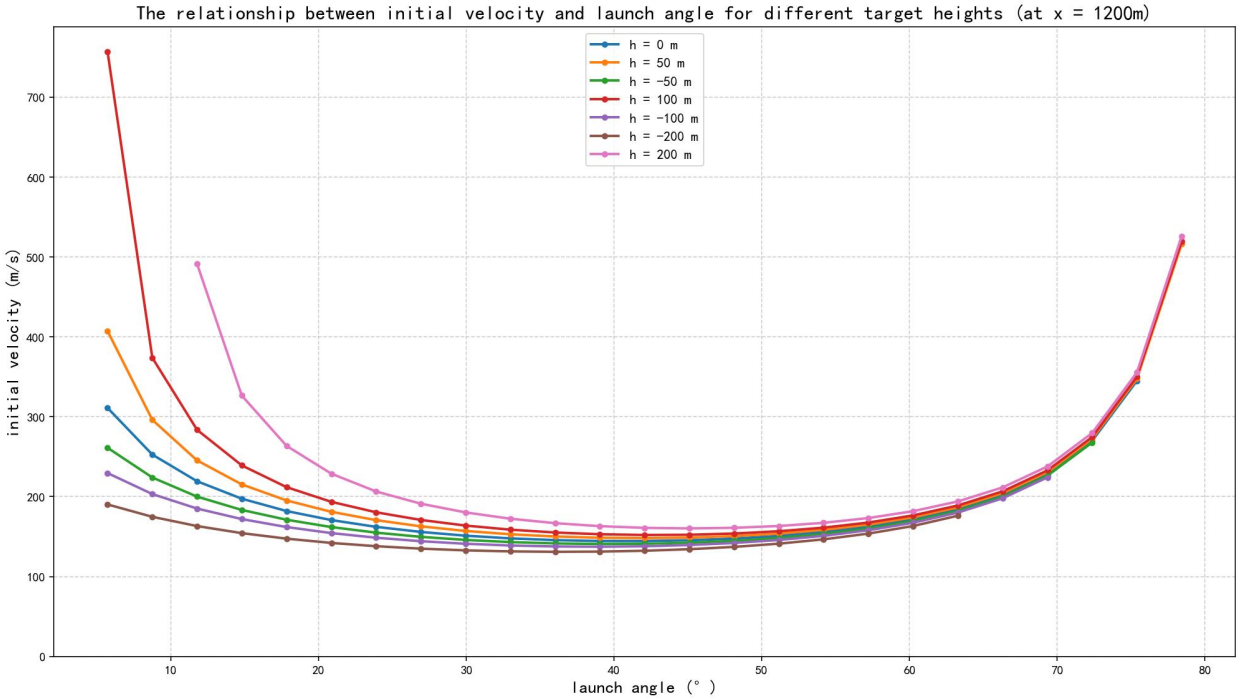


figure 15 Launch Angle vs. Initial Velocity Curves with Different Target Height

After introducing the height difference, the variable that determines the launch angle is no longer x but h. The results of the fitted curve are shown in the figure below:

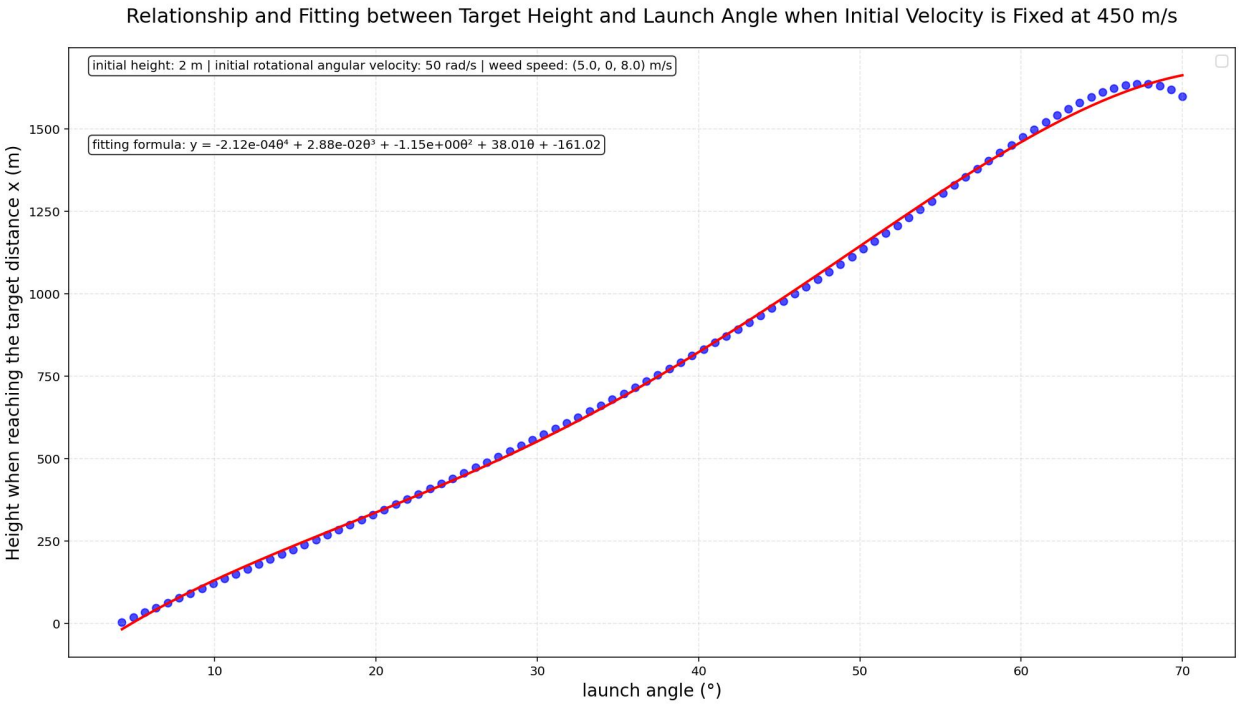


figure 16 The relationship between the launch angle and the height

$$h=-2.12\times10^{-4}\theta_0^{\,4}+2.86\times10^{-2}\theta_0^{\,3}+1.15\times10^{0}\theta_0^{\,2}+38.01\theta_0-161.02$$

(equation 34)

$$\theta_0\approx\frac{h+161.02}{38.01}=\frac{h}{38.01}+4.236 \qquad \text{(equation 35)}$$

For greater rigor, we further analyze the model by introducing different horizontal distances x. We still consider the initial velocity as 450 m/s, and the resulting curve is shown in the figure below:

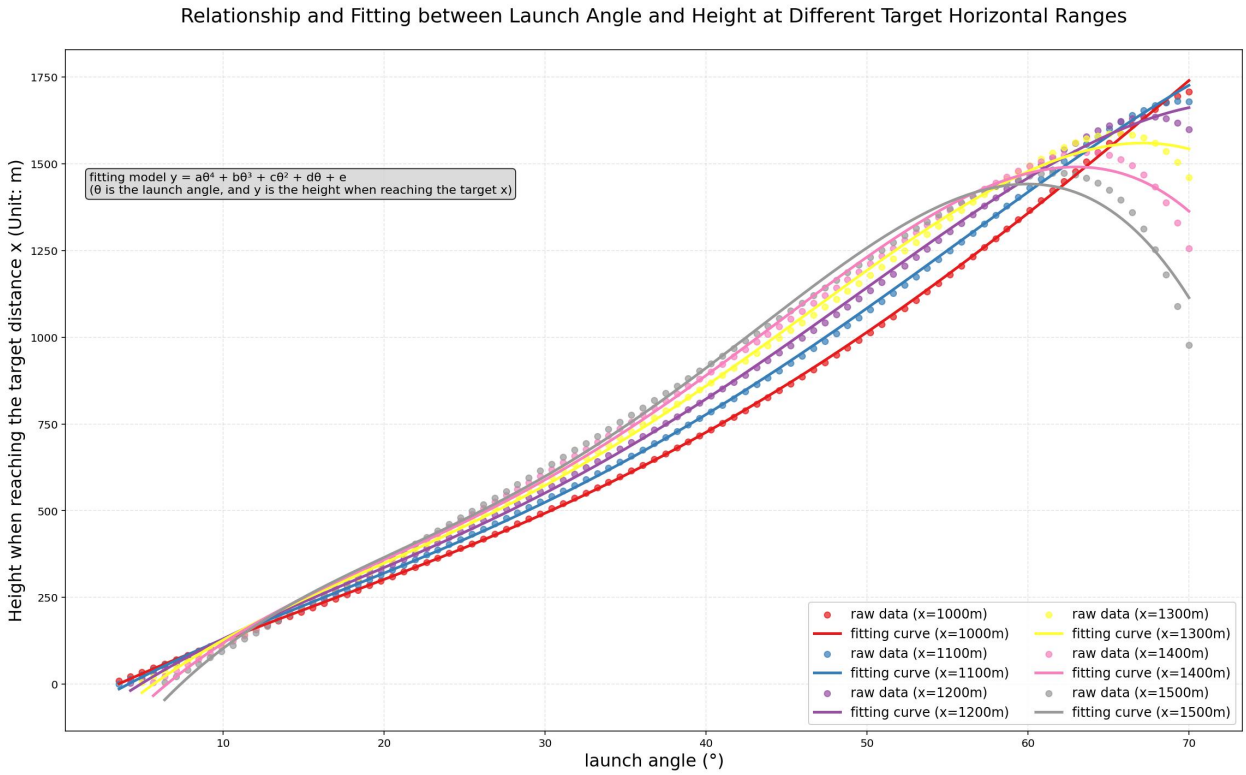


figure 17 The variation of the launch angle when considering both x and h

Considering two variables, the assumed equation is (where h is the height difference):

$$\theta(x,y)=ax^3+bx^2h+cxh^2+dh^3+eh^2+fxh+gh^2+hx+ih+j$$

(equation 36)

The fitted coefficients are as follows:

coefficients	Numerical values（scientific notation）
a	3.666228×10^{-9}
b	1.65455×10^{-8}

coefficients	Numerical values（scientific notation）
c	2.815448×10^{-8}
d	4.2094×10^{-9}
e	-9.66329×10^{-6}
f	-9.14915×10^{-5}
g	$-5.19690-05\times10^{-5}$
h	-1.51163×10^{-2}
i	1.42296×10^{-1}
j	-7.17581×10^{0}

Thus, a more sophisticated formula is obtained. However, this formula involves numerous squared terms and is slow to calculate without a calculator. To address this, we propose two methods:

Method 1: Lookup Table Method
For each specific combination of x (horizontal distance), h (height difference), and wind conditions, a corresponding launch angle is provided. An example is given below, and the complete table can be found in the appendix.

Method 2: Simplified Equation

We propose a simplified linear equation.Meanwhile, we consider the correction for wind speed:

$$\theta = k0 + k1 \cdot x + k2 \cdot h + k3 \cdot ux + k4 \cdot uz \qquad \text{(equation 37)}$$

coefficients	Numerical values（scientific notation）
k0	1.67×10
k1	-7.52×10^{-3}
k2	3.72×10^{-2}
k3	1.54×10^{-2}
k4	6.88×10^{-4}

Given x=1200m, h=450m, ux =6m/s, and uz =9.5m/s:

Predicted angle from the sophisticated model: 25.2594

(absolute error: 0.43°)

Predicted angle from the simplified model: 24.5615

(absolute error: 1.47°)

For large launch angles, this absolute error is negligible. Therefore, the simplified model also maintains sufficient accuracy for practical applications.

5 Conclusion

1..In the horizontal direction, the initial velocity should consistently be set at 450 m/s. The approximate expression for the launch angle is:

$$\theta_0 = \frac{\text{x-286.9091}}{282.32010} \approx \frac{\text{x}}{282} + 1.016$$

2.When altitude is taken into account:

$$v_0(\theta_0, \text{h}) = 0.0010\theta_0^{\,3} + - 0.1026\,\theta_0^{\,2} + \text{C}4.25\theta_0 + (0.0042\text{h}^2 - 0.315\text{h} + 428.7)$$

3.When wind is taken into account:

$$\theta = 1.67 + - 7.52 \times 10 - 3 \cdot \text{x} + 3.72 \times 10 - 2 \cdot \text{h} + 1.54 \times 10 - 2 \cdot \text{ux} + 6.88 \times 10 - 4 \cdot \text{uz}$$

4.When wind is considered, a lateral correction offset must be applied to the launch angle:

$$\beta = \frac{|\vec{u}|}{5} \frac{0.0006\ \theta\ _0^{\ 3} - 0.072\ \theta\ _0^{\ 2} + 2.85\ \theta\ _0 + 2.15}{\text{x}}$$

6 Advantages and Disadvantages Analysis

6.1 Advantages

① The study provides a relatively comprehensive theoretical analysis along with explanations consistent with practical intuition.

② The model is simplified where possible and offers multiple reference curves for artillery officers, significantly reducing computational time.

6.2 Disadvantages

① The differential equations primarily admit only numerical solutions. Although cross-verification has been conducted, analytical solutions are lacking for further validation.

② A quantized analysis approach is adopted, providing curves for various scenarios. However, explicit equations are not introduced for complex conditions. This limitation stems from the constraint that artillery officers lack computational tools, and even if equations were provided, they would be too complex to solve numerically in a short time on the battlefield.

References

[1] 大沽口炮台[M/OL]//维基百科，自由的百科全书. (2025-09-21)[2025-11-09]. https://zh.wikipedia.org/w/index.php?title=%E5%A4%A7%E6%B2%BD%E5%8F%A3%E7%82%AE%E5%8F%B0&oldid=89227376.

[2] WARBURTON R D H, WANG J, BURGD&OUMLRFER J. Analytic Approximations of Projectile Motion with Quadratic Air Resistance[J/OL]. Journal of Service Science and Management, 2010, 03(01): 98-105. DOI:10.4236/jssm.2010.31012.

[3] CHUDINOV P S. Approximate Analytical Investigation of Projectile Motion in a Medium with Quadratic Drag Force[J].

[4] mori.bz.it/Balistica/Mc Coy Modern Exterior Ballistic.pdf?utm_source=chatgpt.com[Z/OL]. [2025-11-08]. https://www.mori.bz.it/Balistica/Mc%20Coy%20Modern%20Exterior%20Ballistic.pdf?utm_source=chatgpt.com.

[5] 师夷长技：清朝在鸦片战争后的炮兵升级尝试[EB/OL]. [2025-11-09]. https://m.thepaper.cn/baijiahao_14606350.

[6] MAXA J, ŠABACKÁ P, BAYER R, 等. The Tuning of a CFD Model for External Ballistics, Followed by Analyses of the Principal Influences on the Drag Coefficient of the .223 Rem Caliber[J/OL]. Technologies, 2025, 13(5): 190. DOI:10.3390/technologies13050190.

[7] WALTERS S J, TURNER R J, FORBES L K. A COMPARISON OF EXPLICIT RUNGE–KUTTA METHODS[J/OL]. The ANZIAM Journal, 2022, 64(3): 227-249. DOI:10.1017/S1446181122000141.

[8] PRICE R H, ROMANO J D. Aim high and go far—Optimal projectile launch angles greater than 45°[J/OL]. American Journal of Physics, 1998, 66(2): 109-113. DOI:10.1119/1.18804.

[9] PADMANABHAN M, ALEEM M. ELEVATIONS FOR MAXIMUM RANGES OF PROJECTILES[C/OL]//31st international symposium on ballistics, vol. 1. exterior ballistics ...: 31st international symposium on ballistics, 4-8 November 2019, Hyderabad, India. Hyderabad, Invalid Date: 782-790[2025-11-09]. https://d.wanfangdata.com.cn/conference/0bd007b74d2e29366b8e92e87d35054a.

[10] 栾永超, 张斌, 李宸凯, 等. 强烟雾干扰下的枪口子弹初速测量方法[J]. 兵工学报, 2025, 46(3): 191-201.

Appendices

Appendices 1

AI Usage Report

Our team used AI for certain repetitive programming and translate tasks, while the problem-solving approaches were independently developed by us without any assistance from AI. The AI primarily utilizes Doubao and GPT-4.

Appendices 2

Introduction：Relationship between Initial Velocity and Angle (for Different Cd Values)

```
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt

# --- 物理参数 ---
g = 9.8           # 重力加速度 (m/s²)
x_target = 1200   # 目标水平距离 (m)
l = 2             # 初始高度参数 (m)，初始高度为 l*sin(theta)
m = 5             # 物体质量 (kg)
r = 0.055         # 物体半径 (m)
rho = 1.175       # 空气密度 (kg/m³)

# --- 计算配置 ---
# 定义 5 个不同的阻力系数 Cd 值
Cd_values = [0.47, 0.5, 0.59, 0.65, 0.75]
# 发射角范围（0° 到 80°，避免极端角度）
theta_deg = np.linspace(0, 77.1, 30)
```

```
# 存储不同Cd值对应的初速度解
all_v0_solutions = []

# --- 主要计算部分 ---
for Cd in Cd_values:
    k = 0.5 * Cd * rho * np.pi * r**2
    v0_solutions = []

    for theta in theta_deg:
        theta_rad = np.radians(theta)
        h0 = l * np.sin(theta_rad)

        def objective(v0):
            v0x = v0 * np.cos(theta_rad)
            v0y = v0 * np.sin(theta_rad)
            initial_state = [0, h0, v0x, v0y]

            def dynamics(state, t):
                x, y, vx, vy = state
                v = np.sqrt(vx**2 + vy**2)
                ax = -(k/m) * v * vx
                ay = -g - (k/m) * v * vy
                return [vx, vy, ax, ay]

            # 预估飞行时间
            t_fall_est = (v0y + np.sqrt(v0y**2 + 2*g*h0)) / g if v0y > 0 else
np.sqrt(2*h0/g)
            t = np.linspace(0, t_fall_est * 1.5, 2000)

            trajectory = odeint(dynamics, initial_state, t)
            y_pos = trajectory[:, 1]
            x_pos = trajectory[:, 0]

            landing_mask = np.where(y_pos <= 0)[0]
            if len(landing_mask) == 0:
```

```
                return 1e10
            else:
                landing_idx = landing_mask[0]
                actual_x = x_pos[landing_idx]
                return actual_x - x_target

        # 求解 v0
        v_low, v_high = 100, 450
        try:
            sol = root_scalar(objective, bracket=[v_low, v_high], method='bisect')
            v0_solutions.append(sol.root)
        except:
            v0_solutions.append(np.nan)

    all_v0_solutions.append(v0_solutions)

# --- 绘图部分 ---
plt.figure(figsize=(12, 7))

# --- 1. 绘制主图：v0 vs theta ---
ax1 = plt.gca() # 获取当前轴
markers = ['o', 's', '^', 'D', 'p']
colors = plt.cm.viridis(np.linspace(0, 1, len(Cd_values))) # 使用颜色映射

# 存储绘图线对象，用于图例
lines1 = []
for i, (Cd, color) in enumerate(zip(Cd_values, colors)):
    v0_solutions = np.array(all_v0_solutions[i])
    valid_mask = ~np.isnan(v0_solutions)
    theta_valid = theta_deg[valid_mask]
    v0_valid = v0_solutions[valid_mask]

    if len(theta_valid) > 0:
        # 特别标记标准值 Cd=0.5
        linestyle = '-' if Cd != 0.5 else '--'
        linewidth = 2.5 if Cd == 0.5 else 2
```

```
        line, = ax1.plot(theta_valid, v0_valid, f'{markers[i]}{linestyle}',
                         color=color,      linewidth=linewidth,      markersize=6,
label=f'Cd = {Cd}')
        lines1.append(line)

ax1.set_xlabel('Launch Angle $\\theta$ (° )', fontsize=12)
ax1.set_ylabel('Initial Velocity $v_0$ (m/s)', color='black', fontsize=12)
ax1.tick_params(axis='y', labelcolor='black')
ax1.set_title(f'Relationship  Between  Initial  Velocity  and  Launch  Angle,  and
Percentage Difference from the Standard Value (Cd=0.5)', fontsize=14)
ax1.grid(True, linestyle='--', alpha=0.7)

# --- 2. 绘制副图：百分差 vs theta ---
ax2 = ax1.twinx() # 创建共享 X 轴的第二个 Y 轴

# 找到标准值 Cd=0.5 的数据索引和对应的 v0 数组
base_Cd = 0.5
base_index = Cd_values.index(base_Cd)
base_v0_solutions = np.array(all_v0_solutions[base_index])

lines2 = []
for i, (Cd, color, marker) in enumerate(zip(Cd_values, colors, markers)):
    if Cd == base_Cd:
        continue # 跳过标准值本身

    v0_solutions = np.array(all_v0_solutions[i])

    # 计算百分差，只对有效数据计算
    valid_mask = ~np.isnan(v0_solutions) & ~np.isnan(base_v0_solutions)
    if not np.any(valid_mask):
        continue # 如果没有有效数据点，则跳过

    theta_valid = theta_deg[valid_mask]
    percent_diff = ((v0_solutions[valid_mask] - base_v0_solutions[valid_mask]) /
base_v0_solutions[valid_mask]) * 100
```

```
    # 绘制百分差曲线，使用与主图相同的颜色但不同的线型
    line, = ax2.plot(theta_valid, percent_diff, f'{marker}:',
                     color=color, linewidth=2, markersize=6, label=f'Cd={Cd} vs
Cd={base_Cd}')
    lines2.append(line)

ax2.set_ylabel(f'Percentage Difference from Cd={base_Cd} (%)', color='darkred',
fontsize=12)
ax2.tick_params(axis='y', labelcolor='darkred')
ax2.axhline(y=0, color='gray', linestyle='--') # 添加0%基准线

# --- 3. 合并图例 ---
lines1.extend(lines2)
labels = [line.get_label() for line in lines1]
ax1.legend(lines1, labels, loc='upper center', fontsize=10)

plt.tight_layout() # 调整布局，防止标签重叠
plt.show()
```

Appendices 3

Introduction：Relationship between Initial Velocity and Angle (Including Final Velocity and Motion Time)

```
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt

# 物理参数（可根据实际情况调整）
g = 9.8           # 重力加速度 (m/s²)
x_target = 1200   # 目标水平距离 (m)
l = 2             # 初始高度参数 (m)，初始高度为 l*sin(theta)
m = 5             # 物体质量 (kg)
r = 0.055         # 物体半径 (m)
rho = 1.175       # 空气密度 (kg/m³)
Cd = 0.47         # 阻力系数（球形物体典型值）
k = 0.5 * Cd * rho * np.pi * r**2  # 空气阻力系数简化项
```

```
# 发射角范围（10° 到 80°，避免极端角度）
theta_deg = np.linspace(0, 80, 30)
v0_solutions = []
flight_times = []  # 存储运动总时间
final_kinetics = []  # 存储最终动能

for theta in theta_deg:
    theta_rad = np.radians(theta)
    # 初始高度：h0 = l*sin(theta)
    h0 = l * np.sin(theta_rad)

    # 定义目标函数：f(v0) = 实际水平距离 - 目标距离 1200m
    def objective(v0):
        # 分解初速度
        v0x = v0 * np.cos(theta_rad)
        v0y = v0 * np.sin(theta_rad)
        # 初始状态 [x, y, vx, vy]
        initial_state = [0, h0, v0x, v0y]

        # 定义运动微分方程
        def dynamics(state, t):
            x, y, vx, vy = state
            v = np.sqrt(vx**2 + vy**2)  # 速度大小
            # 加速度（考虑空气阻力）
            ax = -(k/m) * v * vx  # 水平加速度
            ay = -g - (k/m) * v * vy  # 竖直加速度
            return [vx, vy, ax, ay]

        # 预估飞行时间（确保物体落地）
        # 基于竖直方向运动的保守估计（忽略空气阻力）
        t_fall_est = (v0y + np.sqrt(v0y**2 + 2*g*h0)) / g  # 落地时间估算
        t = np.linspace(0, t_fall_est * 1.5, 2000)  # 延长 1.5 倍确保覆盖

        # 求解运动方程
        trajectory = odeint(dynamics, initial_state, t)
        y_pos = trajectory[:, 1]  # 竖直位置
```

```
        x_pos = trajectory[:, 0]  # 水平位置
        vx = trajectory[:, 2]     # 水平速度
        vy = trajectory[:, 3]     # 竖直速度

        # 查找落地时刻（首次 y<=0 的时间）
        landing_mask = np.where(y_pos <= 0)[0]
        if len(landing_mask) == 0:
            # 若未落地，返回正值（提示需要更大 v0）
            return 1e10, 0, 0  # 额外返回 0 作为无效的时间和动能
        else:
            landing_idx = landing_mask[0]
            actual_x = x_pos[landing_idx]
            flight_time = t[landing_idx]  # 运动总时间
            final_v = np.sqrt(vx[landing_idx]**2 + vy[landing_idx]** 2)  # 落地时
速度大小
            final_ke = 0.5 * m * final_v**2  # 最终动能 (J)
            return actual_x - x_target, flight_time, final_ke

    # 搜索初速度 v0 的范围（确保目标函数在区间内变号）
    v_low = 100   # 初始猜测下限
    v_high = 500  # 初始猜测上限
    f_low, _, _ = objective(v_low)
    f_high, _, _ = objective(v_high)

    # 动态调整搜索区间，确保有解
    while np.sign(f_low) == np.sign(f_high):
        v_high *= 1.5  # 扩大上限
        f_high, _, _ = objective(v_high)
        if v_high > 5000:  # 防止无限扩大
            break

    # 求解 v0 并计算对应参数
    if np.sign(f_low) != np.sign(f_high):
        try:
            # 二分法求解方程：objective(v0) = 0
            sol = root_scalar(lambda v: objective(v)[0], bracket=[v_low, v_high],
```

```
method='bisect')
            v0_solutions.append(sol.root)

            # 计算该初速度对应的运动时间和最终动能
            _, flight_time, final_ke = objective(sol.root)
            flight_times.append(flight_time)
            final_kinetics.append(final_ke)
        except:
            v0_solutions.append(np.nan)
            flight_times.append(np.nan)
            final_kinetics.append(np.nan)
    else:
        v0_solutions.append(np.nan)
        flight_times.append(np.nan)
        final_kinetics.append(np.nan)

# 过滤无效解
v0_solutions = np.array(v0_solutions)
flight_times = np.array(flight_times)
final_kinetics = np.array(final_kinetics)
valid_mask = ~np.isnan(v0_solutions) & ~np.isnan(flight_times) &
~np.isnan(final_kinetics)
theta_valid = theta_deg[valid_mask]
v0_valid = v0_solutions[valid_mask]
time_valid = flight_times[valid_mask]
ke_valid = final_kinetics[valid_mask]

# 绘制结果 - 使用双Y轴
if len(theta_valid) > 0:
    fig, ax1 = plt.subplots(figsize=(12, 7))

    # 绘制初速度曲线（左Y轴）
    color = 'tab:blue'
    ax1.set_xlabel('theta $\\theta$ (° )')
    ax1.set_ylabel('initial velocity $v_0$ (m/s)', color=color)
    ax1.plot(theta_valid, v0_valid, 'bo-', linewidth=2, markersize=5,
```

```
label='initial velocity')
    ax1.tick_params(axis='y', labelcolor=color)
    ax1.grid(True, linestyle='--', alpha=0.7)

    # 创建右侧 Y 轴用于显示飞行时间和最终动能
    ax2 = ax1.twinx()
    # 飞行时间（红色）
    color_time = 'tab:red'
    ax2.plot(theta_valid,  time_valid,  'ro-',  linewidth=2,  markersize=5,
label='flight time')
    ax2.set_ylabel('flight time (s) ', color=color_time)  # 共享 Y 轴标签
    ax2.tick_params(axis='y', labelcolor=color_time)

    # 最终动能（绿色，使用副右侧轴）
    ax3 = ax1.twinx()
    color_ke = 'tab:green'
    # 偏移右侧轴位置，避免重叠
    ax3.spines['right'].set_position(('outward', 60))
    ax3.plot(theta_valid, ke_valid, 'go-', linewidth=2, markersize=5, label='final
kinetic energy')
    ax3.tick_params(axis='y', labelcolor=color_ke)
    ax3.set_ylabel('final kinetic energy (J)', color=color_ke)

    # 添加标题和图例
    plt.title('Relationship between initial velocity, flight time, and final
kinetic energy at different launch angles (target distance 1200m, considering air
resistance)')
    lines, labels = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    lines3, labels3 = ax3.get_legend_handles_labels()
    ax1.legend(lines + lines2 + lines3, labels + labels2 + labels3, loc='best')

    fig.tight_layout()  # 调整布局防止标签重叠
    plt.show()
else:
    print("未找到有效解，请调整参数（如增大 l 或减小 x_target）后重试。")
```

Appendices 4

Introduction：Relationship between Initial Velocity and Angle

```
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt

# 物理参数（可根据实际情况调整）
g = 9.8           # 重力加速度 (m/s²)
x_target = 1200   # 目标水平距离 (m)
l = 2             # 初始高度参数 (m)，初始高度为 l*sin(theta)
m = 5             # 物体质量 (kg)
r = 0.055           # 物体半径 (m)
rho = 1.175       # 空气密度 (kg/m³)
Cd = 0.5          # 阻力系数（球形物体典型值）
k = 0.5 * Cd * rho * np.pi * r**2  # 空气阻力系数简化项

# 发射角范围（0° 到 80°，避免极端角度）
theta_deg = np.linspace(0, 77.1, 30)
v0_solutions = []

for theta in theta_deg:
    theta_rad = np.radians(theta)
    # 初始高度：h0 = l*sin(theta)
    h0 = l * np.sin(theta_rad)

    # 定义目标函数：f(v0) = 实际水平距离 - 目标距离 1200m
    def objective(v0):
        # 分解初速度
        v0x = v0 * np.cos(theta_rad)
        v0y = v0 * np.sin(theta_rad)
        # 初始状态 [x, y, vx, vy]
        initial_state = [0, h0, v0x, v0y]

        # 定义运动微分方程
        def dynamics(state, t):
```

```
            x, y, vx, vy = state
            v = np.sqrt(vx**2 + vy**2)  # 速度大小
            # 加速度（考虑空气阻力）
            ax = -(k/m) * v * vx  # 水平加速度
            ay = -g - (k/m) * v * vy  # 竖直加速度
            return [vx, vy, ax, ay]

        # 预估飞行时间（确保物体落地）
        # 基于竖直方向运动的保守估计（忽略空气阻力）
        t_fall_est = (v0y + np.sqrt(v0y**2 + 2*g*h0)) / g  # 落地时间估算
        t = np.linspace(0, t_fall_est * 1.5, 2000)  # 延长 1.5 倍确保覆盖

        # 求解运动方程
        trajectory = odeint(dynamics, initial_state, t)
        y_pos = trajectory[:, 1]  # 竖直位置
        x_pos = trajectory[:, 0]  # 水平位置

        # 查找落地时刻（首次 y<=0 的时间）
        landing_mask = np.where(y_pos <= 0)[0]
        if len(landing_mask) == 0:
            # 若未落地，返回正值（提示需要更大 v0）
            return 1e10
        else:
            landing_idx = landing_mask[0]
            actual_x = x_pos[landing_idx]
            return actual_x - x_target  # 目标函数：误差=实际距离-目标距离

    # 搜索初速度 v0 的范围（确保目标函数在区间内变号）
    v_low = 100   # 初始猜测下限
    v_high = 450  # 初始猜测上限
    f_low = objective(v_low)
    f_high = objective(v_high)

    # 动态调整搜索区间，确保有解
    while np.sign(f_low) == np.sign(f_high):
        v_high *= 1.5  # 扩大上限
```

```
        f_high = objective(v_high)
        if v_high > 5000:  # 防止无限扩大
            break

    # 求解 v0
    if np.sign(f_low) != np.sign(f_high):
        try:
            # 二分法求解方程：objective(v0) = 0
            sol = root_scalar(objective, bracket=[v_low, v_high], method='bisect')
            v0_solutions.append(sol.root)
        except:
            v0_solutions.append(np.nan)
    else:
        v0_solutions.append(np.nan)

# 过滤无效解
v0_solutions = np.array(v0_solutions)
valid_mask = ~np.isnan(v0_solutions)
theta_valid = theta_deg[valid_mask]
v0_valid = v0_solutions[valid_mask]

# 绘制结果
if len(theta_valid) > 0:
    plt.figure(figsize=(10, 6))
    plt.plot(theta_valid, v0_valid, 'bo-', linewidth=2, markersize=5)
    plt.xlabel('theta $\\theta$ (° )')
    plt.ylabel('initial velocity $v_0$ (m/s)')
    plt.title(f'Relationship between $v_0$ and launch angle $\\theta$ for a
horizontal ranger of 1200 meters(considering air resistance)')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.show()
else:
    print("未找到有效解，请调整参数（如增大 l 或减小 x_target）后重试。")
```

Appendices 5

Introduction：Relationship between Initial Velocity and Angular Velocity (under Different Air Densities)

```
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt

# 物理参数（可根据实际情况调整）
g = 9.8           # 重力加速度 (m/s²)
x_target = 1200   # 目标水平距离 (m)
l = 2             # 初始高度参数 (m)，初始高度为 l*sin(theta)
m = 5             # 物体质量 (kg)
r = 0.055         # 物体半径 (m)
rho = 1.175      # 空气密度 (kg/m³)
Cd = 0.47         # 阻力系数（球形物体典型值）
k = 0.5 * Cd * rho * np.pi * r**2  # 空气阻力系数简化项

# 发射角范围（0°到80°，避免极端角度）
theta_deg = np.linspace(0, 77.1, 30)
v0_solutions = []
kinetic_energies = []  # 存储最终动能

for theta in theta_deg:
    theta_rad = np.radians(theta)
    # 初始高度：h0 = l*sin(theta)
    h0 = l * np.sin(theta_rad)

    # 定义目标函数：f(v0) = 实际水平距离 - 目标距离1200m
    def objective(v0):
        # 分解初速度
        v0x = v0 * np.cos(theta_rad)
        v0y = v0 * np.sin(theta_rad)
        # 初始状态 [x, y, vx, vy]
        initial_state = [0, h0, v0x, v0y]

        # 定义运动微分方程
        def dynamics(state, t):
            x, y, vx, vy = state
```

```
            v = np.sqrt(vx**2 + vy**2)  # 速度大小
            # 加速度（考虑空气阻力）
            ax = -(k/m) * v * vx  # 水平加速度
            ay = -g - (k/m) * v * vy  # 竖直加速度
            return [vx, vy, ax, ay]

        # 预估飞行时间（确保物体落地）
        # 基于竖直方向运动的保守估计（忽略空气阻力）
        t_fall_est = (v0y + np.sqrt(v0y**2 + 2*g*h0)) / g  # 落地时间估算
        t = np.linspace(0, t_fall_est * 1.5, 2000)  # 延长 1.5 倍确保覆盖

        # 求解运动方程
        trajectory = odeint(dynamics, initial_state, t)
        y_pos = trajectory[:, 1]  # 竖直位置
        x_pos = trajectory[:, 0]  # 水平位置

        # 查找落地时刻（首次 y<=0 的时间）
        landing_mask = np.where(y_pos <= 0)[0]
        if len(landing_mask) == 0:
            # 若未落地，返回正值（提示需要更大 v0）
            return 1e10
        else:
            landing_idx = landing_mask[0]
            actual_x = x_pos[landing_idx]
            return actual_x - x_target  # 目标函数：误差=实际距离-目标距离

    # 搜索初速度 v0 的范围（确保目标函数在区间内变号）
    v_low = 100   # 初始猜测下限
    v_high = 450  # 初始猜测上限
    f_low = objective(v_low)
    f_high = objective(v_high)

    # 动态调整搜索区间，确保有解
    while np.sign(f_low) == np.sign(f_high):
        v_high *= 1.5  # 扩大上限
        f_high = objective(v_high)
```

```
        if v_high > 5000:  # 防止无限扩大
            break

    # 求解 v0 并计算落地时的动能
    if np.sign(f_low) != np.sign(f_high):
        try:
            # 二分法求解方程：objective(v0) = 0
            sol = root_scalar(objective, bracket=[v_low, v_high], method='bisect')
            v0 = sol.root
            v0_solutions.append(v0)

            # 重新计算轨迹以获取落地时的速度
            v0x = v0 * np.cos(theta_rad)
            v0y = v0 * np.sin(theta_rad)
            initial_state = [0, h0, v0x, v0y]

            def dynamics(state, t):
                x, y, vx, vy = state
                v = np.sqrt(vx**2 + vy**2)
                ax = -(k/m) * v * vx
                ay = -g - (k/m) * v * vy
                return [vx, vy, ax, ay]

            t_fall_est = (v0y + np.sqrt(v0y**2 + 2*g*h0)) / g
            t = np.linspace(0, t_fall_est * 1.5, 2000)
            trajectory = odeint(dynamics, initial_state, t)
            y_pos = trajectory[:, 1]
            landing_mask = np.where(y_pos <= 0)[0]
            landing_idx = landing_mask[0]

            # 获取落地时的速度分量并计算动能 (Ek = 0.5 * m * v²)
            vx_land = trajectory[landing_idx, 2]
            vy_land = trajectory[landing_idx, 3]
            v_land = np.sqrt(vx_land**2 + vy_land**2)
            ek = 0.5 * m * v_land**2
            kinetic_energies.append(ek)
```

```
        except:
            v0_solutions.append(np.nan)
            kinetic_energies.append(np.nan)
    else:
        v0_solutions.append(np.nan)
        kinetic_energies.append(np.nan)

# 过滤无效解
v0_solutions = np.array(v0_solutions)
kinetic_energies = np.array(kinetic_energies)
valid_mask = ~np.isnan(v0_solutions) & ~np.isnan(kinetic_energies)
theta_valid = theta_deg[valid_mask]
v0_valid = v0_solutions[valid_mask]
ke_valid = kinetic_energies[valid_mask]

# 绘制结果
if len(theta_valid) > 0:
    fig, ax1 = plt.subplots(figsize=(10, 6))

    # 绘制初速度曲线（左侧纵轴）
    color = 'tab:blue'
    ax1.set_xlabel('theta $\\theta$ (° )')
    ax1.set_ylabel('initial velocity $v_0$ (m/s)', color=color)
    ax1.plot(theta_valid,    v0_valid,    'bo-',    linewidth=2,    markersize=5,
label='initial velocity')
    ax1.tick_params(axis='y', labelcolor=color)
    ax1.grid(True, linestyle='--', alpha=0.7)

    # 创建右侧纵轴并绘制动能曲线
    ax2 = ax1.twinx()
    color = 'tab:red'
    ax2.set_ylabel('final kinetic energy (J)', color=color)
    ax2.plot(theta_valid,    ke_valid,    'ro--',    linewidth=2,    markersize=5,
label='final kinetic energy')
    ax2.tick_params(axis='y', labelcolor=color)
```

```
    # 添加标题和图例
    plt.title('The Relationship Between Initial Velocity and Landing Kinetic Energy at Different Launch Angles (Range of 1200 Meters, Considering Air Resistance)')
    lines1, labels1 = ax1.get_legend_handles_labels()
    lines2, labels2 = ax2.get_legend_handles_labels()
    ax1.legend(lines1 + lines2, labels1 + labels2, loc='best')

    fig.tight_layout()  # 调整布局
    plt.show()
else:
    print("未找到有效解，请调整参数（如增大 l 或减小 x_target）后重试。")
```

Appendices 6

Introductin：Relationship between Initial Velocity and Angular Velocity (under Different Air Densities, with Error Bars)

```
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt

# 定义计算不同 rho 值下 v0 与 theta 关系的函数（与之前相同，此处省略）
def calculate_v0_theta(rho_value):
    g = 9.8           # 重力加速度 (m/s²)
    x_target = 1200   # 目标水平距离 (m)
    l = 2             # 初始高度参数 (m)
    m = 5             # 物体质量 (kg)
    r = 0.055         # 物体半径 (m)
    Cd = 0.47         # 阻力系数
    k = 0.5 * Cd * rho_value * np.pi * r**2  # 空气阻力系数

    theta_deg = np.linspace(0, 80, 30)
    v0_solutions = []

    for theta in theta_deg:
        theta_rad = np.radians(theta)
```

```
        h0 = l * np.sin(theta_rad)

        def objective(v0):
            v0x = v0 * np.cos(theta_rad)
            v0y = v0 * np.sin(theta_rad)
            initial_state = [0, h0, v0x, v0y]

            def dynamics(state, t):
                x, y, vx, vy = state
                v = np.sqrt(vx**2 + vy**2)
                ax = -(k/m) * v * vx
                ay = -g - (k/m) * v * vy
                return [vx, vy, ax, ay]

            t_fall_est = (v0y + np.sqrt(v0y**2 + 2*g*h0)) / g
            t = np.linspace(0, t_fall_est * 1.5, 2000)
            trajectory = odeint(dynamics, initial_state, t)
            y_pos = trajectory[:, 1]
            x_pos = trajectory[:, 0]

            landing_mask = np.where(y_pos <= 0)[0]
            if len(landing_mask) == 0:
                return 1e10
            else:
                landing_idx = landing_mask[0]
                actual_x = x_pos[landing_idx]
                return actual_x - x_target

        v_low = 100
        v_high = 450
        f_low = objective(v_low)
        f_high = objective(v_high)

        while np.sign(f_low) == np.sign(f_high):
            v_high *= 1.5
            f_high = objective(v_high)
```

```
            if v_high > 5000:
                break

        if np.sign(f_low) != np.sign(f_high):
            try:
                sol = root_scalar(objective, bracket=[v_low, v_high],
method='bisect')
                v0_solutions.append(sol.root)
            except:
                v0_solutions.append(np.nan)
        else:
            v0_solutions.append(np.nan)

    v0_solutions = np.array(v0_solutions)
    valid_mask = ~np.isnan(v0_solutions)
    return theta_deg[valid_mask], v0_solutions[valid_mask]

# 定义参数和样式
rho_values = [0.937, 1.013, 1.065, 1.120, 1.175]
styles = [('b', 'o'), ('g', 's'), ('r', '^'), ('c', 'd'), ('y', 'x')]

# 计算所有结果
results = {}
for rho in rho_values:
    theta, v0 = calculate_v0_theta(rho)
    results[rho] = (theta, v0)

# 获取标准曲线数据
ref_rho = 1.175
ref_theta, ref_v0 = results[ref_rho]

# 创建图形和坐标轴
fig, ax1 = plt.subplots(figsize=(12, 7))

# 绘制主曲线
for i, rho in enumerate(rho_values):
```

```
    theta, v0 = results[rho]
    if len(theta) > 0:
        color, marker = styles[i]
        ax1.plot(theta, v0, f'{color}-{marker}',
                 linewidth=2, markersize=5,
                 label=f'rho = {rho} kg/m³')

# 设置主坐标轴
ax1.set_xlabel('theta $\\theta$ (° )', fontsize=12)
ax1.set_ylabel('initial velocity $v_0$ (m/s)', fontsize=12)
ax1.set_title('Relationship between initial velocity and launch angle (with
percentage difference)', fontsize=14)
ax1.grid(True, linestyle='--', alpha=0.7)

# 调整主图例位置到左侧空白区域（坐标系统：(0,0)左下角，(1,1)右上角）
ax1.legend(title='air density', fontsize=10,
           loc='upper left', bbox_to_anchor=(0, 0.72))

# 创建右侧副坐标轴
ax2 = ax1.twinx()
ax2.set_ylabel('Percentage Difference (%)', fontsize=12, color='purple')

# 计算并绘制百分差
for i, rho in enumerate(rho_values):
    if rho == ref_rho:
        continue

    theta, v0 = results[rho]
    color, marker = styles[i]
    valid_theta = []
    percent_diff = []

    for t, v in zip(theta, v0):
        if ref_theta.min() <= t <= ref_theta.max():
            ref_v = np.interp(t, ref_theta, ref_v0)
            diff = (v - ref_v) / ref_v * 100
```

```
            valid_theta.append(t)
            percent_diff.append(diff)

    if valid_theta:
        ax2.plot(valid_theta, percent_diff, f'{color}--', linewidth=1.5,
                 label=f'{rho} vs {ref_rho}')

# 调整副图例位置到左侧下方空白区域
ax2.legend(title='Percentage Difference', fontsize=8,
           loc='upper left', bbox_to_anchor=(0, 0.85))

ax2.tick_params(axis='y', labelcolor='purple')
plt.tight_layout()
plt.show()
```

Appendices 7

Introduction：Relationship between Initial Velocity and Angular Velocity (Including Maximum Height)

```
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import root_scalar
import matplotlib.pyplot as plt

# 物理参数（可根据实际情况调整）
g = 9.8           # 重力加速度 (m/s²)
x_target = 1200   # 目标水平距离 (m)
l = 2             # 初始高度参数 (m)，初始高度为 l*sin(theta)
m = 5             # 物体质量 (kg)
r = 0.055         # 物体半径 (m)
rho = 1.175      # 空气密度 (kg/m³)
Cd = 0.47         # 阻力系数（球形物体典型值）
k = 0.5 * Cd * rho * np.pi * r**2  # 空气阻力系数简化项

# 发射角范围（0°到80°，避免极端角度）
theta_deg = np.linspace(0, 77.1, 30)
v0_solutions = []
max_heights = []  # 存储每个角度对应的最大高度
```

```
for theta in theta_deg:
    theta_rad = np.radians(theta)
    h0 = l * np.sin(theta_rad)  # 初始高度

    # 定义目标函数：f(v0) = 实际水平距离 - 目标距离1200m
    def objective(v0):
        v0x = v0 * np.cos(theta_rad)
        v0y = v0 * np.sin(theta_rad)
        initial_state = [0, h0, v0x, v0y]

        # 定义运动微分方程
        def dynamics(state, t):
            x, y, vx, vy = state
            v = np.sqrt(vx**2 + vy**2)
            ax = -(k/m) * v * vx
            ay = -g - (k/m) * v * vy
            return [vx, vy, ax, ay]

        # 预估飞行时间
        t_fall_est = (v0y + np.sqrt(v0y**2 + 2*g*h0)) / g
        t = np.linspace(0, t_fall_est * 1.5, 2000)

        # 求解运动方程
        trajectory = odeint(dynamics, initial_state, t)
        y_pos = trajectory[:, 1]
        x_pos = trajectory[:, 0]

        # 查找落地时刻
        landing_mask = np.where(y_pos <= 0)[0]
        if len(landing_mask) == 0:
            return 1e10
        else:
            landing_idx = landing_mask[0]
            actual_x = x_pos[landing_idx]
            return actual_x - x_target
```

```
# 搜索初速度 v0 的范围
v_low = 100
v_high = 450
f_low = objective(v_low)
f_high = objective(v_high)

# 动态调整搜索区间
while np.sign(f_low) == np.sign(f_high):
    v_high *= 1.5
    f_high = objective(v_high)
    if v_high > 5000:
        break

# 求解 v0 并计算最大高度
if np.sign(f_low) != np.sign(f_high):
    try:
        sol = root_scalar(objective, bracket=[v_low, v_high], method='bisect')
        v0_solutions.append(sol.root)

        # 计算该初速度下的最大高度
        v0x = sol.root * np.cos(theta_rad)
        v0y = sol.root * np.sin(theta_rad)
        initial_state = [0, h0, v0x, v0y]

        # 重新计算轨迹以获取最大高度
        def dynamics_max(state, t):
            x, y, vx, vy = state
            v = np.sqrt(vx**2 + vy**2)
            ax = -(k/m) * v * vx
            ay = -g - (k/m) * v * vy
            return [vx, vy, ax, ay]

        t_fall_est = (v0y + np.sqrt(v0y**2 + 2*g*h0)) / g
        t = np.linspace(0, t_fall_est * 1.5, 2000)
        trajectory = odeint(dynamics_max, initial_state, t)
```

```
            y_pos = trajectory[:, 1]
            max_height = np.max(y_pos)  # 找到最大高度
            max_heights.append(max_height)

        except:
            v0_solutions.append(np.nan)
            max_heights.append(np.nan)
    else:
        v0_solutions.append(np.nan)
        max_heights.append(np.nan)

# 过滤无效解
v0_solutions = np.array(v0_solutions)
max_heights = np.array(max_heights)
valid_mask = ~np.isnan(v0_solutions)
theta_valid = theta_deg[valid_mask]
v0_valid = v0_solutions[valid_mask]
heights_valid = max_heights[valid_mask]

# 绘制结果并添加标注
if len(theta_valid) > 0:
    plt.figure(figsize=(12, 7))
    plt.plot(theta_valid, v0_valid, 'bo-', linewidth=2, markersize=5)

    # 为每个点添加最大高度标注
    for i in range(len(theta_valid)):
        plt.annotate(f'h={heights_valid[i]:.1f}m',
                     (theta_valid[i], v0_valid[i]),
                     textcoords="offset points",
                     xytext=(0, 10),  # 文本相对于点的位置
                     ha='center',
                     fontsize=8,
                     bbox=dict(boxstyle="round,pad=0.3",   fc="white",   ec="gray",
alpha=0.7))

    plt.xlabel('theta $\\theta$ (° )')
```

```
    plt.ylabel('initial velocity $v_0$ (m/s)')
    plt.title(f'v0 vs theta for 1200m range (with max height annotations)')
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.tight_layout()  # 调整布局防止标注被截断
    plt.show()
else:
    print("未找到有效解，请调整参数（如增大 l 或减小 x_target）后重试。")
```

Appendices 8

Introduction：Relationship between Angle and Initial Velocity (Including Running Time and Final Kinetic Energy)

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# ====================== 物理参数设置（适配新条件）======================
rho = 1.175       # 空气密度, kg/m^3
r = 0.055          # 球半径, m
m = 5            # 球质量, kg
g = 9.8          # 重力加速度, m/s^2
Cd = 0.4          # 空气阻力系数
Cm = 0.2          # 马格努斯力系数
Cw = 0.015        # 自转气动阻尼系数（推荐值）
uwz = 8.0         # z 轴方向风速, m/s（正=沿 z 轴正方向，负=反方向）
l_sin_theta = 2  # 初始竖直高度, m（lsinθ）
target_x = 1200   # 目标水平射程, m
omega0 = 150      # 初始自转角速度（沿 x 轴）, rad/s（线膛火炮场景>0）

# 派生参数
A = np.pi * r**2          # 迎风面积, m^2
I = (2/5) * m * r**2      # 转动惯量, kg·m^2
v0_range = (80, 1000)      # 初速度搜索范围, m/s
tol = 0.5                 # 射程误差容忍度, m

# ====================== 三维耦合微分方程组（修正马格努斯力）======================
def ode_3d(t, y, v0, theta):
    """三维平移-自转耦合微分方程：y = [vx, vy, vz, omega, x, y, z]"""
```

```
    vx, vy, vz, omega, x, y_pos, z = y

    # 1. 相对速度（风力仅沿 z 轴）
    v_rel_x = vx
    v_rel_y = vy
    v_rel_z = vz - uwz
    V = np.sqrt(v_rel_x**2 + v_rel_y**2 + v_rel_z**2)
    if V < 1e-6:
        V = 1e-6

    # 2. 空气阻力
    Fdx = -0.5 * Cd * rho * A * V * v_rel_x
    Fdy = -0.5 * Cd * rho * A * V * v_rel_y
    Fdz = -0.5 * Cd * rho * A * V * v_rel_z

    # 3. 马格努斯力（核心修正：补充 × r）
    Fmx = 0.0  # x 轴无马格努斯力
    Fmy = -0.5 * rho * A * Cm * omega * r * v_rel_z
    Fmz = 0.5 * rho * A * Cm * omega * r * v_rel_y

    # 4. 平移加速度
    ax = (Fdx + Fmx) / m
    ay = (Fdy + Fmy - m * g) / m
    az = (Fdz + Fmz) / m

    # 5. 自转角加速度
    alpha_omega = (-0.5 * Cw * rho * A * r**2 * V * omega) / I

    # 6. 位置变化率
    dx = vx
    dy = vy
    dz_dt = vz

    return [ax, ay, az, alpha_omega, dx, dy, dz_dt]

# ==================== 轨迹仿真与初速度求解 ====================
```

```
def simulate_3d(v0, theta):
    """仿真三维轨迹，返回落地时的 x 轴位移、运行时间和最终动能"""
    # 初始状态：[vx0, vy0, vz0, omega0, x0, y0, z0]
    vx0 = v0 * np.cos(theta)
    vy0 = v0 * np.sin(theta)
    vz0 = 0.0  # 初始侧向速度为 0
    y0 = [vx0, vy0, vz0, omega0, 0.0, l_sin_theta, 0.0]

    # 落地事件：y_pos=0 时终止仿真
    def land_event(t, y):
        return y[5]  # y[5]是竖直位置 y_pos
    land_event.terminal = True  # 触发事件后终止
    land_event.direction = -1   # 仅当 y_pos 从正变负时触发（落地）

    # 求解微分方程（RK45 算法）
    sol = solve_ivp(
        fun=lambda t, y: ode_3d(t, y, v0, theta),
        t_span=[0.0, 200.0],  # 最大仿真时间（足够落地）
        y0=y0,
        events=land_event,
        method='RK45',
        rtol=1e-6,
        atol=1e-9
    )

    # 若未落地（无事件触发），返回无效值
    if len(sol.t_events[0]) == 0:
        return np.nan, np.nan, np.nan

    # 落地时的状态
    t_land = sol.t_events[0][0]  # 落地时间
    x_land = sol.y[4, -1]        # 射程
    vx_final = sol.y[0, -1]      # 最终 x 方向速度
    vy_final = sol.y[1, -1]      # 最终 y 方向速度
    vz_final = sol.y[2, -1]      # 最终 z 方向速度
    omega_final = sol.y[3, -1]   # 最终角速度
```

```
    # 计算最终动能（平动动能+转动动能）
    ke_translation = 0.5 * m * (vx_final**2 + vy_final**2 + vz_final**2)
    ke_rotation = 0.5 * I * omega_final**2
    ke_total = ke_translation + ke_rotation

    return x_land, t_land, ke_total

def find_v0_3d(theta):
    """二分法求解满足目标射程的初速度 v0，同时返回运行时间和动能"""
    v_low, v_high = v0_range
    # 迭代搜索（误差小于 tol 时停止）
    for _ in range(50):  # 最多迭代 50 次，确保收敛
        v_mid = (v_low + v_high) / 2
        x_land, _, _ = simulate_3d(v_mid, theta)

        if np.isnan(x_land):
            # 未落地，需要更大初速度
            v_low = v_mid
        elif x_land < target_x:
            # 射程不足，增大初速度
            v_low = v_mid
        else:
            # 射程超了，减小初速度
            v_high = v_mid

        if v_high - v_low < tol:
            break

    # 计算最终的运行时间和动能
    _, t_land, ke_total = simulate_3d(v_high, theta)
    return v_high, t_land, ke_total

# ====================== 遍历发射角，计算对应参数 ======================
# 发射角范围：0~80 度（转换为弧度），取 30 个点
thetas_deg = np.linspace(0.5, 80.0, 30)  # 避免 0 度（无竖直速度）
```

```
thetas_rad = np.radians(thetas_deg)

# 计算每个角度对应的初速度、运行时间和动能
v0_list = []
time_list = []
ke_list = []

for theta_rad, theta_deg in zip(thetas_rad, thetas_deg):
    v0, t_land, ke = find_v0_3d(theta_rad)
    v0_list.append(v0)
    time_list.append(t_land)
    ke_list.append(ke)
    print(f"theta：{theta_deg:.1f}°  → initial velocity：{v0:.1f} m/s, runtime ：
{t_land:.1f} s, final kinetic energy：{ke:.1f} J")

# ====================== 绘制多曲线关系图（双纵轴）======================
fig, ax1 = plt.subplots(figsize=(12, 6))

# 左侧纵轴：初速度
color = '#2E86AB'
ax1.set_xlabel('theta(° )', fontsize=12)
ax1.set_ylabel('initial velocity (m/s)', color=color, fontsize=12)
ax1.plot(thetas_deg,  v0_list,  'o-',  color=color,  linewidth=2,  markersize=6,
label='initial velocity')
ax1.tick_params(axis='y', labelcolor=color)
ax1.grid(True, alpha=0.3, linestyle='--')
ax1.set_xlim(0, 85)

# 创建右侧纵轴
ax2 = ax1.twinx()
color = '#E74C3C'
ax2.set_ylabel('runtime (s)', color=color, fontsize=12)  # 第一个右侧变量
ax2.plot(thetas_deg,  time_list,  's-',  color=color,  linewidth=2,  markersize=6,
label='runtime ')
ax2.tick_params(axis='y', labelcolor=color)
```

```
# 创建第二个右侧纵轴
ax3 = ax1.twinx()
# 调整第二个右侧轴的位置，避免重叠
ax3.spines['right'].set_position(('outward', 60))
color = '#27AE60'
ax3.set_ylabel('final kinetic energy (J)', color=color, fontsize=12)  # 第二个右侧变量
ax3.plot(thetas_deg, ke_list, '^-', color=color, linewidth=2, markersize=6, label='final kinetic energy')
ax3.tick_params(axis='y', labelcolor=color)

# 整合图例
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
lines3, labels3 = ax3.get_legend_handles_labels()
ax1.legend(lines + lines2 + lines3, labels + labels2 + labels3, loc='upper left')

# 标题和参数标注
plt.title('Relationship Between Initial Velocity, Runtime, Final Kinetic Energy and Launch Angle', fontsize=14, pad=20)
param_text = f'Z-axis wind speed={uwz} m/s\n initial rotational angular velocity={omega0} rad/s\n target range={target_x} m'
plt.text(0.45, 0.98, param_text, transform=ax1.transAxes,
         verticalalignment='top', fontsize=10,
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))

plt.tight_layout()
plt.show()
```

Appendices 9

Introduction：Relationship between Angle and Initial Velocity (Solution Combinations)

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# ==================== 物理参数设置（与原代码保持一致）====================
```

```
rho = 1.175        # 空气密度, kg/m^3
r = 0.055           # 球半径, m
m = 5             # 球质量, kg
g = 9.8           # 重力加速度, m/s^2
Cd = 0.4          # 空气阻力系数
Cm = 0.2          # 马格努斯力系数
Cw = 0.015        # 自转气动阻尼系数
uwz = 8.0         # z 轴方向风速, m/s
l_sin_theta = 2  # 初始竖直高度, m
target_x = 1200   # 目标水平射程, m
omega0 = 150      # 初始自转角速度, rad/s

# 派生参数
A = np.pi * r**2          # 迎风面积, m^2
I = (2/5) * m * r**2      # 转动惯量, kg·m^2
tol = 0.1                 # 角度搜索精度, 度

# ====================== 三维耦合微分方程组（保持不变）======================
def ode_3d(t, y, v0, theta):
    vx, vy, vz, omega, x, y_pos, z = y

    # 相对速度
    v_rel_x = vx
    v_rel_y = vy
    v_rel_z = vz - uwz
    V = np.sqrt(v_rel_x**2 + v_rel_y**2 + v_rel_z**2)
    if V < 1e-6:
        V = 1e-6

    # 空气阻力
    Fdx = -0.5 * Cd * rho * A * V * v_rel_x
    Fdy = -0.5 * Cd * rho * A * V * v_rel_y
    Fdz = -0.5 * Cd * rho * A * V * v_rel_z

    # 马格努斯力
    Fmx = 0.0
```

```
    Fmy = -0.5 * rho * A * Cm * omega * r * v_rel_z
    Fmz = 0.5 * rho * A * Cm * omega * r * v_rel_y

    # 平移加速度
    ax = (Fdx + Fmx) / m
    ay = (Fdy + Fmy - m * g) / m
    az = (Fdz + Fmz) / m

    # 自转角加速度
    alpha_omega = (-0.5 * Cw * rho * A * r**2 * V * omega) / I

    # 位置变化率
    dx = vx
    dy = vy
    dz_dt = vz

    return [ax, ay, az, alpha_omega, dx, dy, dz_dt]

# ==================== 轨迹仿真函数（保持不变）====================
def simulate_3d(v0, theta):
    vx0 = v0 * np.cos(theta)
    vy0 = v0 * np.sin(theta)
    vz0 = 0.0
    y0 = [vx0, vy0, vz0, omega0, 0.0, l_sin_theta, 0.0]

    def land_event(t, y):
        return y[5]
    land_event.terminal = True
    land_event.direction = -1

    sol = solve_ivp(
        fun=lambda t, y: ode_3d(t, y, v0, theta),
        t_span=[0.0, 200.0],
        y0=y0,
        events=land_event,
        method='RK45',
```

```
        rtol=1e-6,
        atol=1e-9
    )

    if len(sol.t_events[0]) == 0:
        return np.nan
    return sol.y[4, -1]

# ====================== 搜索指定初速度对应的角度（新增功能）======================
def find_theta_for_v0(v0_target, angle_range_deg):
    """在指定角度范围内搜索对应目标初速度的发射角"""
    theta_low_deg, theta_high_deg = angle_range_deg
    theta_low = np.radians(theta_low_deg)
    theta_high = np.radians(theta_high_deg)

    # 二分法搜索
    for _ in range(100):  # 增加迭代次数以提高精度
        theta_mid = (theta_low + theta_high) / 2
        x_land = simulate_3d(v0_target, theta_mid)

        if np.isnan(x_land):
            # 未落地，需要增大角度
            theta_low = theta_mid
        elif x_land < target_x:
            # 射程不足，需要增大角度
            theta_low = theta_mid
        else:
            # 射程超了，需要减小角度
            theta_high = theta_mid

        # 转换为度判断精度
        if np.degrees(theta_high - theta_low) < tol:
            break

    return np.degrees((theta_low + theta_high) / 2)
```

```
# ==================== 主程序执行 ====================
if __name__ == "__main__":
    # 搜索角度范围：0.5 度到 10 度（满足角度小于 10 度的条件）
    angle_range = (0.5, 10.0)
    target_v0 = 450  # 目标初速度，m/s

    # 计算对应的发射角
    result_theta = find_theta_for_v0(target_v0, angle_range)

    # 验证结果
    verify_x = simulate_3d(target_v0, np.radians(result_theta))
    print(f"在角度小于 10 度时，初始速度为 450m/s 对应的发射角为：{result_theta:.4f}
度")
    print(f"验证：该角度下的射程为：{verify_x:.2f}m（目标射程：{target_x}m）")
```

Appendices 10

Introduction：Relationship between Angle and Initial Velocity (at Different Air Densities)

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# ==================== 物理参数设置（适配新条件）====================
# 设置 5 个不同的空气密度值 (kg/m^3)
rho_values = [0.880, 0.956, 1.046, 1.112, 1.175]  # 包含原有的 1.175 作为参考
r = 0.055           # 球半径，m
m = 5               # 球质量，kg
g = 9.8             # 重力加速度，m/s^2
Cd = 0.4            # 空气阻力系数
Cm = 0.2            # 马格努斯力系数
Cw = 0.015          # 自转气动阻尼系数
uwz = 8.0           # z 轴方向风速，m/s
l_sin_theta = 2     # 初始竖直高度，m（lsinθ）
target_x = 1200     # 目标水平射程，m
omega0 = 150        # 初始自转角速度（沿 x 轴），rad/s

# 派生参数
```

```
A = np.pi * r**2          # 迎风面积, m^2
I = (2/5) * m * r**2      # 转动惯量, kg·m^2
v0_range = (80, 1000)     # 初速度搜索范围, m/s
tol = 0.5                 # 射程误差容忍度, m

# ==================== 三维耦合微分方程组（修正马格努斯力）====================
def ode_3d(t, y, v0, theta, rho):  # 新增 rho 参数
    """三维平移-自转耦合微分方程：y = [vx, vy, vz, omega, x, y, z]"""
    vx, vy, vz, omega, x, y_pos, z = y

    # 1. 相对速度（风力仅沿 z 轴）
    v_rel_x = vx
    v_rel_y = vy
    v_rel_z = vz - uwz
    V = np.sqrt(v_rel_x**2 + v_rel_y**2 + v_rel_z**2)
    if V < 1e-6:
        V = 1e-6

    # 2. 空气阻力
    Fdx = -0.5 * Cd * rho * A * V * v_rel_x
    Fdy = -0.5 * Cd * rho * A * V * v_rel_y
    Fdz = -0.5 * Cd * rho * A * V * v_rel_z

    # 3. 马格努斯力（包含球半径 r）
    Fmx = 0.0  # x 轴无马格努斯力
    Fmy = -0.5 * rho * A * Cm * omega * r * v_rel_z
    Fmz = 0.5 * rho * A * Cm * omega * r * v_rel_y

    # 4. 平移加速度
    ax = (Fdx + Fmx) / m
    ay = (Fdy + Fmy - m * g) / m
    az = (Fdz + Fmz) / m

    # 5. 自转角加速度
    alpha_omega = (-0.5 * Cw * rho * A * r**2 * V * omega) / I
```

```
    # 6. 位置变化率
    dx = vx
    dy = vy
    dz_dt = vz

    return [ax, ay, az, alpha_omega, dx, dy, dz_dt]

# ==================== 轨迹仿真与初速度求解 ====================
def simulate_3d(v0, theta, rho):  # 新增 rho 参数
    """仿真三维轨迹，返回落地时的 x 轴位移（射程）"""
    # 初始状态：[vx0, vy0, vz0, omega0, x0, y0, z0]
    vx0 = v0 * np.cos(theta)
    vy0 = v0 * np.sin(theta)
    vz0 = 0.0  # 初始侧向速度为 0
    y0 = [vx0, vy0, vz0, omega0, 0.0, l_sin_theta, 0.0]

    # 落地事件：y_pos=0 时终止仿真
    def land_event(t, y):
        return y[5]  # y[5]是竖直位置 y_pos
    land_event.terminal = True  # 触发事件后终止
    land_event.direction = -1   # 仅当 y_pos 从正变负时触发（落地）

    # 求解微分方程（RK45 算法）
    sol = solve_ivp(
        fun=lambda t, y: ode_3d(t, y, v0, theta, rho),  # 传入 rho
        t_span=[0.0, 200.0],  # 最大仿真时间（足够落地）
        y0=y0,
        events=land_event,
        method='RK45',
        rtol=1e-6,
        atol=1e-9
    )

    # 若未落地（无事件触发），返回无效值
    if len(sol.t_events[0]) == 0:
        return np.nan
```

```
    # 返回落地时的 x 轴位移（射程）
    return sol.y[4, -1]

def find_v0_3d(theta, rho):  # 新增 rho 参数
    """二分法求解满足目标射程的初速度 v0"""
    v_low, v_high = v0_range
    # 迭代搜索（误差小于 tol 时停止）
    for _ in range(50):  # 最多迭代 50 次，确保收敛
        v_mid = (v_low + v_high) / 2
        x_land = simulate_3d(v_mid, theta, rho)  # 传入 rho

        if np.isnan(x_land):
            # 未落地，需要更大初速度
            v_low = v_mid
        elif x_land < target_x:
            # 射程不足，增大初速度
            v_low = v_mid
        else:
            # 射程超了，减小初速度
            v_high = v_mid

        if v_high - v_low < tol:
            break
    return v_high

# ====================== 遍历发射角和不同 rho 值，计算对应初速度 ======================
# 发射角范围：0~80 度（转换为弧度），取 30 个点
thetas_deg = np.linspace(0.5, 80.0, 30)  # 避免 0 度（无竖直速度）
thetas_rad = np.radians(thetas_deg)

# 存储不同 rho 值对应的初速度列表
all_v0_lists = []
for rho in rho_values:
    print(f"正在计算空气密度 rho = {rho} kg/m^3 时的初速度...")
    v0_list = []
```

```
    for theta_rad, theta_deg in zip(thetas_rad, thetas_deg):
        v0 = find_v0_3d(theta_rad, rho)  # 传入 rho
        v0_list.append(v0)
        print(f"  theta: {theta_deg:.1f}°  → initial velocity: {v0:.1f} m/s")
    all_v0_lists.append(v0_list)

# ====================== 绘制多条初速度-发射角关系曲线 ======================
plt.figure(figsize=(12, 6))
# 定义不同曲线的样式（颜色和标记）
colors = ['#2E86AB', '#E74C3C', '#27AE60', '#F39C12', '#9B59B6']
markers = ['o', 's', '^', 'D', 'p']

for i, (rho, v0_list) in enumerate(zip(rho_values, all_v0_lists)):
    plt.plot(thetas_deg, v0_list,
             marker=markers[i],
             color=colors[i],
             linewidth=2,
             markersize=6,
             label=f'rho = {rho} kg/m³')

plt.xlabel('theta (° )', fontsize=12)
plt.ylabel('initial velocity (m/s)', fontsize=12)
plt.title('Relationship Between Initial Velocity and Launch Angle Under Different
Air Densities (Including Z-axis Wind + X-axis Rotation)', fontsize=14, pad=20)
plt.grid(True, alpha=0.3, linestyle='--')
plt.xlim(0, 85)
# 自动调整 y 轴范围，留出 5%的余量
all_v0 = [v for sublist in all_v0_lists for v in sublist]
plt.ylim(min(all_v0)*0.95, max(all_v0)*1.05)

# 标注关键参数
param_text = f'Z-axis wind speed={uwz} m/s\n initial rotational angular
velocity={omega0} rad/s\n target range={target_x} m'
plt.text(0.02, 0.98, param_text, transform=plt.gca().transAxes,
         verticalalignment='top', fontsize=10,
         bbox=dict(boxstyle='round', facecolor='white', alpha=0.8))
```

```
plt.legend(fontsize=10)  # 添加图例
plt.tight_layout()
plt.show()
```

Appendices 11

Introduction：Relationship between Angle and Initial Velocity (Considering Wind Speeds in the x and z Directions)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import warnings
warnings.filterwarnings('ignore')

# ---------------------------- 1. 核心参数定义 ----------------------------
l = 2                # 初始高度相关参数 (m)，根据实际场景调整
g = 9.8                # 重力加速度 (m/s²)
m = 5                # 物体质量 (kg)，根据实际场景调整
r = 0.055                # 物体半径 (m)，根据实际场景调整
rho = 1.175            # 空气密度 (kg/m³)
C_d = 0.5             # 阻力系数 (球体默认 0.47)
u_x = 5.0              # x 方向风速 (m/s)，可调整
u_z = 2.0              # z 方向风速 (m/s)，可调整
k = 0.5 * C_d * rho * np.pi * r**2  # 阻力系数

# ---------------------------- 2. 运动微分方程 ----------------------------
def motion(t, state):
    x, y, z, vx, vy, vz = state
    # 相对风速
    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    # 加速度计算
    ax = - (k / m) * v_rel * vrelx if v_rel > 1e-6 else 0.0
    ay = -g - (k / m) * v_rel * vrely if v_rel > 1e-6 else -g
```

```
    az = - (k / m) * v_rel * vrelz if v_rel > 1e-6 else 0.0
    return [vx, vy, vz, ax, ay, az]

# --------------------------- 3. 落地判定函数 ---------------------------
def hit_ground(t, state):
    return state[1]
hit_ground.terminal = True
hit_ground.direction = -1

# --------------------------- 4. 目标函数与 v0 求解 ---------------------------
def objective(v0, theta):
    x0 = 0.0
    y0 = 1 * np.sin(theta)
    z0 = 0.0
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    v0z = 0.0
    initial_state = [x0, y0, z0, v0x, v0y, v0z]
    t_span = [0, 10000]
    sol = solve_ivp(
        fun=motion, t_span=t_span, y0=initial_state,
        events=hit_ground, method='RK45', rtol=1e-6, atol=1e-9
    )
    if len(sol.t_events[0]) == 0:
        return 1e10
    x_landing = sol.y_events[0][0, 0]
    return x_landing - 1200

def find_v0(theta, v0_min=1.0, v0_max=1000.0, tol=1e-3, max_iter=100):
    f_min = objective(v0_min, theta)
    f_max = objective(v0_max, theta)
    # 调整搜索范围
    if f_min * f_max > 0:
        while f_min * f_max > 0 and v0_max < 1e5:
            v0_max *= 2
            f_max = objective(v0_max, theta)
```

```
        if f_min * f_max > 0:
            return None, None  # 同时返回 v0 和 z_landing（无解时均为 None）
    # 二分迭代
    for _ in range(max_iter):
        v0_mid = (v0_min + v0_max) / 2
        f_mid = objective(v0_mid, theta)
        if abs(f_mid) < tol:
            # 计算对应 z_landing
            initial_state = [0.0, l*np.sin(theta), 0.0, v0_mid*np.cos(theta),
v0_mid*np.sin(theta), 0.0]
            sol = solve_ivp(fun=motion, t_span=[0,100], y0=initial_state,
events=hit_ground, method='RK45')
            z_landing = sol.y_events[0][0, 2]
            return v0_mid, z_landing
        if f_min * f_mid < 0:
            v0_max = v0_mid
            f_max = f_mid
        else:
            v0_min = v0_mid
            f_min = f_mid
    # 迭代结束后计算 z_landing
    initial_state = [0.0, l*np.sin(theta), 0.0, ((v0_min+v0_max)/2)*np.cos(theta),
((v0_min+v0_max)/2)*np.sin(theta), 0.0]
    sol = solve_ivp(fun=motion, t_span=[0,100], y0=initial_state,
events=hit_ground, method='RK45')
    z_landing = sol.y_events[0][0, 2]
    return (v0_min + v0_max) / 2, z_landing

# -------------------------- 5. 遍历 theta 求解+结果可视化 ------------------------
--
theta_list = np.linspace(0, 80 * np.pi / 180, 30)  # 减少点数避免标注拥挤（30 个
点）
theta_deg = theta_list * 180 / np.pi
v0_list = []
z_landing_list = []
```

```
# 求解 v0 和对应的 z_landing
for theta in theta_list:
    v0, z_land = find_v0(theta)
    if v0 is not None and z_land is not None:
        v0_list.append(v0)
        z_landing_list.append(z_land)
    else:
        v0_list.append(np.nan)
        z_landing_list.append(np.nan)

# -------------------------- 6. 绘制 v0-theta 图（带 z 值标注） ----------------------
------
fig, ax = plt.subplots(1, 1, figsize=(12, 8))

# 绘制 v0-theta 曲线
ax.plot(theta_deg, v0_list, 'b-o', linewidth=2, markersize=6, label='v0-theta
relationship')

# 为每个有效点添加 z 值标注
for i in range(len(theta_deg)):
    if not np.isnan(v0_list[i]) and not np.isnan(z_landing_list[i]):
        # 标注格式：z=xx.xx m，位置偏移避免重叠
        ax.annotate(
            f'z={z_landing_list[i]:.2f}m',
            xy=(theta_deg[i], v0_list[i]),
            xytext=(5, 5),  # 文字相对于点的偏移（x+5, y+5 像素）
            textcoords='offset points',
            fontsize=8,
            color='darkred',
            bbox=dict(boxstyle='round,pad=0.2', facecolor='white', alpha=0.7)   #
白色背景提高可读性
        )

# 图表美化
ax.set_xlabel('theta θ (°)', fontsize=14)
ax.set_ylabel('initial velocity v₀ (m/s)', fontsize=14)
```

```
ax.set_title('The relationship between initial velocity and launch angle (noting the z-direction distance at landing)', fontsize=16)
ax.grid(True, alpha=0.3)
ax.legend(fontsize=12)

plt.tight_layout()
plt.show()
```

Appendices 12

Introduction：Relationship between Angle and Initial Velocity (Solution Combinations)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import warnings
warnings.filterwarnings('ignore')

l = 2
g = 9.8
m = 5
r = 0.055
rho = 1.175
C_d = 0.47
u_x = 5.0
u_z = 2.0
k = 0.5 * C_d * rho * np.pi * r**2

def motion(t, state):
    x, y, z, vx, vy, vz = state
    # 相对风速
    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    # 加速度计算
    ax = - (k / m) * v_rel * vrelx if v_rel > 1e-6 else 0.0
    ay = -g - (k / m) * v_rel * vrely if v_rel > 1e-6 else -g
```

```
    az = - (k / m) * v_rel * vrelz if v_rel > 1e-6 else 0.0
    return [vx, vy, vz, ax, ay, az]


def hit_ground(t, state):
    return state[1]
hit_ground.terminal = True
hit_ground.direction = -1

def get_x_landing(v0, theta):
    x0 = 0.0
    y0 = l * np.sin(theta)
    z0 = 0.0
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    v0z = 0.0
    initial_state = [x0, y0, z0, v0x, v0y, v0z]
    t_span = [0, 10000]
    sol = solve_ivp(
        fun=motion, t_span=t_span, y0=initial_state,
        events=hit_ground, method='RK45', rtol=1e-6, atol=1e-9
    )
    if len(sol.t_events[0]) == 0:
        return None  # 未落地
    return sol.y_events[0][0, 0]

def find_theta_for_v0(v0, target_x=1200, max_deg=10, tol=1e-4, max_iter=100):
    # 转换角度为弧度
    theta_min = 0  # 最小角度（弧度）
    theta_max = max_deg * np.pi / 180  # 最大角度（弧度）

    # 计算边界值
    x_min = get_x_landing(v0, theta_min)
    x_max = get_x_landing(v0, theta_max)

    # 检查解是否存在
```

```
    if x_min is None or x_max is None:
        return None
    if (x_min - target_x) * (x_max - target_x) > 0:
        # 两端点值在目标值同侧，无解
        return None

    # 二分法求解
    for _ in range(max_iter):
        theta_mid = (theta_min + theta_max) / 2
        x_mid = get_x_landing(v0, theta_mid)

        if abs(x_mid - target_x) < tol:
            return theta_mid * 180 / np.pi  # 转换为度

        if (x_mid - target_x) * (x_min - target_x) < 0:
            theta_max = theta_mid
            x_max = x_mid
        else:
            theta_min = theta_mid
            x_min = x_mid

    # 返回最接近的解
    return (theta_min + theta_max) / 2 * 180 / np.pi

v0_target = 450  # 目标初速度
theta_result = find_theta_for_v0(v0_target)

if theta_result is not None and theta_result <= 10:
    print(f"当初速度为{int(v0_target)}m/s时，小于10度的角度为：{theta_result:.4f}度")
    # 验证结果
    x_verify = get_x_landing(v0_target, theta_result * np.pi / 180)
    print(f"验证：该角度下的落地x坐标为：{x_verify:.2f}m（目标：1200m）")
else:
    print(f"在初速度为{int(v0_target)}m/s时，未找到小于10度且能使落地x坐标接近1200m的角度")
```

```
theta_list = np.linspace(0, 80 * np.pi / 180, 30)  # 减少点数避免标注拥挤（30 个点）
theta_deg = theta_list * 180 / np.pi
v0_list = []
z_landing_list = []

# 求解 v0 和对应的 z_landing
def objective(v0, theta):
    x0 = 0.0
    y0 = l * np.sin(theta)
    z0 = 0.0
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    v0z = 0.0
    initial_state = [x0, y0, z0, v0x, v0y, v0z]
    t_span = [0, 10000]
    sol = solve_ivp(
        fun=motion, t_span=t_span, y0=initial_state,
        events=hit_ground, method='RK45', rtol=1e-6, atol=1e-9
    )
    if len(sol.t_events[0]) == 0:
        return 1e10
    x_landing = sol.y_events[0][0, 0]
    return x_landing - 1200

def find_v0(theta, v0_min=1.0, v0_max=1000.0, tol=1e-3, max_iter=100):
    f_min = objective(v0_min, theta)
    f_max = objective(v0_max, theta)
    # 调整搜索范围
    if f_min * f_max > 0:
        while f_min * f_max > 0 and v0_max < 1e5:
            v0_max *= 2
            f_max = objective(v0_max, theta)
        if f_min * f_max > 0:
            return None, None  # 同时返回 v0 和 z_landing（无解时均为 None）
```

```
    # 二分迭代
    for _ in range(max_iter):
        v0_mid = (v0_min + v0_max) / 2
        f_mid = objective(v0_mid, theta)
        if abs(f_mid) < tol:
            # 计算对应 z_landing
            initial_state = [0.0, l*np.sin(theta), 0.0, v0_mid*np.cos(theta),
v0_mid*np.sin(theta), 0.0]
            sol = solve_ivp(fun=motion, t_span=[0,100], y0=initial_state,
events=hit_ground, method='RK45')
            z_landing = sol.y_events[0][0, 2]
            return v0_mid, z_landing
        if f_min * f_mid < 0:
            v0_max = v0_mid
            f_max = f_mid
        else:
            v0_min = v0_mid
            f_min = f_mid
    # 迭代结束后计算 z_landing
    initial_state = [0.0, l*np.sin(theta), 0.0, ((v0_min+v0_max)/2)*np.cos(theta),
((v0_min+v0_max)/2)*np.sin(theta), 0.0]
    sol = solve_ivp(fun=motion, t_span=[0,100], y0=initial_state,
events=hit_ground, method='RK45')
    z_landing = sol.y_events[0][0, 2]
    return (v0_min + v0_max) / 2, z_landing

for theta in theta_list:
    v0, z_land = find_v0(theta)
    if v0 is not None and z_land is not None:
        v0_list.append(v0)
        z_landing_list.append(z_land)
    else:
        v0_list.append(np.nan)
        z_landing_list.append(np.nan)

fig, ax = plt.subplots(1, 1, figsize=(12, 8))
```

```
# 绘制 v0-theta 曲线
ax.plot(theta_deg, v0_list, 'b-o', linewidth=2, markersize=6, label='v0-theta relationship')

# 标记我们找到的解
if theta_result is not None and theta_result <= 10:
    ax.plot(theta_result, v0_target, 'rs', markersize=10, label=f'v0=450m/s at θ={theta_result:.2f}°')

# 为每个有效点添加 z 值标注
for i in range(len(theta_deg)):
    if not np.isnan(v0_list[i]) and not np.isnan(z_landing_list[i]):
        ax.annotate(
            f'z={z_landing_list[i]:.2f}m',
            xy=(theta_deg[i], v0_list[i]),
            xytext=(5, 5),
            textcoords='offset points',
            fontsize=8,
            color='darkred',
            bbox=dict(boxstyle='round,pad=0.2', facecolor='white', alpha=0.7)
        )

# 图表美化
ax.set_xlabel('theta θ (°)', fontsize=14)
ax.set_ylabel('initial velocity v₀ (m/s)', fontsize=14)
ax.set_title('The relationship between initial velocity and launch angle (noting the z-direction distance at landing)', fontsize=16)
ax.grid(True, alpha=0.3)
ax.legend(fontsize=12)

plt.tight_layout()
plt.show()
```

Appendices 13

Introduction：Relationship between Angle and Initial Velocity (Motion Time and Final Kinetic Energy)

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import warnings
warnings.filterwarnings('ignore')

# ---------------------------- 1. 核心参数定义 ----------------------------
l = 2                # 初始高度相关参数 (m)，根据实际场景调整
g = 9.8                # 重力加速度 (m/s²)
m = 5                # 物体质量 (kg)，根据实际场景调整
r = 0.055                # 物体半径 (m)，根据实际场景调整
rho = 1.175            # 空气密度 (kg/m³)
C_d = 0.47             # 阻力系数 (球体默认 0.47)
u_x = 5.0              # x 方向风速 (m/s)，可调整
u_z = 2.0              # z 方向风速 (m/s)，可调整
k = 0.5 * C_d * rho * np.pi * r**2  # 阻力系数

# ---------------------------- 2. 运动微分方程 ----------------------------
def motion(t, state):
    x, y, z, vx, vy, vz = state
    # 相对风速
    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    # 加速度计算
    ax = - (k / m) * v_rel * vrelx if v_rel > 1e-6 else 0.0
    ay = -g - (k / m) * v_rel * vrely if v_rel > 1e-6 else -g
    az = - (k / m) * v_rel * vrelz if v_rel > 1e-6 else 0.0
    return [vx, vy, vz, ax, ay, az]

# ---------------------------- 3. 落地判定函数 ----------------------------
def hit_ground(t, state):
    return state[1]
hit_ground.terminal = True
hit_ground.direction = -1
```

```
# ------------------------- 4. 目标函数与 v0 求解 -------------------------
def objective(v0, theta):
    x0 = 0.0
    y0 = l * np.sin(theta)
    z0 = 0.0
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    v0z = 0.0
    initial_state = [x0, y0, z0, v0x, v0y, v0z]
    t_span = [0, 10000]
    sol = solve_ivp(
        fun=motion, t_span=t_span, y0=initial_state,
        events=hit_ground, method='RK45', rtol=1e-6, atol=1e-9
    )
    if len(sol.t_events[0]) == 0:
        return 1e10
    x_landing = sol.y_events[0][0, 0]
    return x_landing - 1200

def find_v0(theta, v0_min=1.0, v0_max=1000.0, tol=1e-3, max_iter=100):
    f_min = objective(v0_min, theta)
    f_max = objective(v0_max, theta)
    # 调整搜索范围
    if f_min * f_max > 0:
        while f_min * f_max > 0 and v0_max < 1e5:
            v0_max *= 2
            f_max = objective(v0_max, theta)
        if f_min * f_max > 0:
            return None, None, None  # 增加返回运动时间
    # 二分迭代
    for _ in range(max_iter):
        v0_mid = (v0_min + v0_max) / 2
        f_mid = objective(v0_mid, theta)
        if abs(f_mid) < tol:
            # 计算对应 z_landing 和运动时间
            initial_state = [0.0, l*np.sin(theta), 0.0, v0_mid*np.cos(theta),
```

```
v0_mid*np.sin(theta), 0.0]
            sol = solve_ivp(fun=motion, t_span=[0,100], y0=initial_state,
events=hit_ground, method='RK45')
            z_landing = sol.y_events[0][0, 2]
            t_landing = sol.t_events[0][0]  # 运动时间
            # 计算落地时的速度
            vx_final = sol.y[3][-1]
            vy_final = sol.y[4][-1]
            vz_final = sol.y[5][-1]
            v_final = np.sqrt(vx_final**2 + vy_final**2 + vz_final**2)
            kinetic_energy = 0.5 * m * v_final**2  # 最终动能
            return v0_mid, kinetic_energy, t_landing
        if f_min * f_mid < 0:
            v0_max = v0_mid
            f_max = f_mid
        else:
            v0_min = v0_mid
            f_min = f_mid
    # 迭代结束后计算相关参数
    v0 = (v0_min + v0_max) / 2
    initial_state = [0.0, l*np.sin(theta), 0.0, v0*np.cos(theta),
v0*np.sin(theta), 0.0]
    sol = solve_ivp(fun=motion, t_span=[0,100], y0=initial_state,
events=hit_ground, method='RK45')
    t_landing = sol.t_events[0][0] if len(sol.t_events[0]) > 0 else np.nan
    # 计算落地时的动能
    if len(sol.y[3]) > 0 and len(sol.t_events[0]) > 0:
        vx_final = sol.y[3][-1]
        vy_final = sol.y[4][-1]
        vz_final = sol.y[5][-1]
        v_final = np.sqrt(vx_final**2 + vy_final**2 + vz_final**2)
        kinetic_energy = 0.5 * m * v_final**2
    else:
        kinetic_energy = np.nan
    return v0, kinetic_energy, t_landing
```

```
# ------------------------- 5. 遍历 theta 求解+结果可视化 -----------------------
--
theta_list = np.linspace(0, 80 * np.pi / 180, 30)  # 30 个点
theta_deg = theta_list * 180 / np.pi
v0_list = []
kinetic_energy_list = []  # 最终动能列表
time_list = []  # 运动时间列表

# 求解 v0、最终动能和运动时间
for theta in theta_list:
    v0, ke, t = find_v0(theta)
    if v0 is not None and ke is not None and t is not None:
        v0_list.append(v0)
        kinetic_energy_list.append(ke)
        time_list.append(t)
    else:
        v0_list.append(np.nan)
        kinetic_energy_list.append(np.nan)
        time_list.append(np.nan)

# --------------------------- 6. 绘制双坐标轴图像 ---------------------------
fig, ax1 = plt.subplots(1, 1, figsize=(12, 8))

# 左侧坐标轴：初速度
color = 'tab:blue'
ax1.set_xlabel('theta θ (° )', fontsize=14)
ax1.set_ylabel('initial velocity v₀ (m/s)', color=color, fontsize=14)
ax1.plot(theta_deg, v0_list, 'b-o', linewidth=2, markersize=6, label='Initial
velocity')
ax1.tick_params(axis='y', labelcolor=color)
ax1.grid(True, alpha=0.3)
ax1.legend(loc='upper left', fontsize=12)

# 创建右侧坐标轴
ax2 = ax1.twinx()
color = 'tab:red'
```

```
ax2.set_ylabel('Kinetic Energy (J)', color=color, fontsize=14)  # 动能单位为焦耳
ax2.plot(theta_deg,  kinetic_energy_list,  'r-s',  linewidth=2,  markersize=6,
label='Final Kinetic Energy')
ax2.tick_params(axis='y', labelcolor=color)

# 再创建一个右侧坐标轴用于显示运动时间
ax3 = ax1.twinx()
color = 'tab:green'
# 调整右侧坐标轴位置，避免重叠
ax3.spines['right'].set_position(('outward', 60))
ax3.set_ylabel('Motion Time (s)', color=color, fontsize=14)  # 时间单位为秒
ax3.plot(theta_deg,  time_list,  'g-^',  linewidth=2,  markersize=6,  label='Motion
Time')
ax3.tick_params(axis='y', labelcolor=color)

# 合并图例
lines1, labels1 = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
lines3, labels3 = ax3.get_legend_handles_labels()
ax1.legend(lines1 + lines2 + lines3, labels1 + labels2 + labels3, loc='upper
center', fontsize=12)

ax1.set_title('Relationship between launch angle and initial velocity, final
kinetic energy, motion time', fontsize=16)
plt.tight_layout()
plt.show()
```

Appendices 14

Introductin： Relationship between Angle and Initial Velocity (at Different Air Densities)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import warnings
warnings.filterwarnings('ignore')

# -------------------------- 1. 核心参数定义 --------------------------
```

```
l = 2                # 初始高度相关参数 (m)，根据实际场景调整
g = 9.8              # 重力加速度 (m/s²)
m = 5                # 物体质量 (kg)，根据实际场景调整
r = 0.055            # 物体半径 (m)，根据实际场景调整
# 定义 5 个不同的空气密度值
rho_values = [0.909, 1.023, 1.056, 1.124, 1.175]  # 新增：5 个空气密度值
C_d = 0.47           # 阻力系数 (球体默认 0.47)
u_x = 5.0            # x 方向风速 (m/s)，可调整
u_z = 2.0            # z 方向风速 (m/s)，可调整

# --------------------------- 2. 运动微分方程 ---------------------------
def motion(t, state, rho):  # 新增：接收 rho 参数
    x, y, z, vx, vy, vz = state
    k = 0.5 * C_d * rho * np.pi * r**2  # 阻力系数（移到此处，随 rho 变化）
    # 相对风速
    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    # 加速度计算
    ax = - (k / m) * v_rel * vrelx if v_rel > 1e-6 else 0.0
    ay = -g - (k / m) * v_rel * vrely if v_rel > 1e-6 else -g
    az = - (k / m) * v_rel * vrelz if v_rel > 1e-6 else 0.0
    return [vx, vy, vz, ax, ay, az]

# --------------------------- 3. 落地判定函数 ---------------------------
def hit_ground(t, state):
    return state[1]
hit_ground.terminal = True
hit_ground.direction = -1

# --------------------------- 4. 目标函数与 v0 求解 ---------------------------
def objective(v0, theta, rho):  # 新增：接收 rho 参数
    x0 = 0.0
    y0 = l * np.sin(theta)
    z0 = 0.0
```

```
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    v0z = 0.0
    initial_state = [x0, y0, z0, v0x, v0y, v0z]
    t_span = [0, 10000]
    # 求解时传入当前 rho 值
    sol = solve_ivp(
        fun=lambda t, state: motion(t, state, rho),
        t_span=t_span,
        y0=initial_state,
        events=hit_ground,
        method='RK45',
        rtol=1e-6,
        atol=1e-9
    )
    if len(sol.t_events[0]) == 0:
        return 1e10
    x_landing = sol.y_events[0][0, 0]
    return x_landing - 1200

def find_v0(theta, rho, v0_min=1.0, v0_max=1000.0, tol=1e-3, max_iter=100):  # 新
增：接收 rho 参数
    f_min = objective(v0_min, theta, rho)
    f_max = objective(v0_max, theta, rho)
    # 调整搜索范围
    if f_min * f_max > 0:
        while f_min * f_max > 0 and v0_max < 1e5:
            v0_max *= 2
            f_max = objective(v0_max, theta, rho)
        if f_min * f_max > 0:
            return None, None  # 同时返回 v0 和 z_landing（无解时均为 None）
    # 二分迭代
    for _ in range(max_iter):
        v0_mid = (v0_min + v0_max) / 2
        f_mid = objective(v0_mid, theta, rho)
        if abs(f_mid) < tol:
```

```
            # 计算对应 z_landing
            initial_state  =  [0.0,  l*np.sin(theta),  0.0,  v0_mid*np.cos(theta),
v0_mid*np.sin(theta), 0.0]
            sol = solve_ivp(
                fun=lambda t, state: motion(t, state, rho),
                t_span=[0,100],
                y0=initial_state,
                events=hit_ground,
                method='RK45'
            )
            z_landing = sol.y_events[0][0, 2]
            return v0_mid, z_landing
        if f_min * f_mid < 0:
            v0_max = v0_mid
            f_max = f_mid
        else:
            v0_min = v0_mid
            f_min = f_mid
    # 迭代结束后计算 z_landing
    initial_state = [0.0, l*np.sin(theta), 0.0, ((v0_min+v0_max)/2)*np.cos(theta),
((v0_min+v0_max)/2)*np.sin(theta), 0.0]
    sol = solve_ivp(
        fun=lambda t, state: motion(t, state, rho),
        t_span=[0,100],
        y0=initial_state,
        events=hit_ground,
        method='RK45'
    )
    z_landing = sol.y_events[0][0, 2]
    return (v0_min + v0_max) / 2, z_landing

# -------------------------- 5. 遍历 theta 和 rho 求解 --------------------------
theta_list = np.linspace(0, 80 * np.pi / 180, 30)  # 角度范围
theta_deg = theta_list * 180 / np.pi

# 存储不同 rho 对应的结果
```

```
results = []
for rho in rho_values:
    v0_list = []
    for theta in theta_list:
        v0, _ = find_v0(theta, rho)  # 不需要 z 值，用_接收
        if v0 is not None:
            v0_list.append(v0)
        else:
            v0_list.append(np.nan)
    results.append(v0_list)

# ---------------------------- 6. 绘制多条关系线 ----------------------------
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
colors = ['blue', 'green', 'red', 'purple', 'orange']  # 为每条线分配颜色
markers = ['o', 's', '^', 'D', 'v']  # 不同标记

# 绘制每条曲线
for i, rho in enumerate(rho_values):
    ax.plot(
        theta_deg,
        results[i],
        color=colors[i],
        marker=markers[i],
        markersize=5,
        linewidth=2,
        label=f'rho = {rho} kg/m³'
    )

# 图表美化
ax.set_xlabel('theta θ (°)', fontsize=14)
ax.set_ylabel('initial velocity v₀ (m/s)', fontsize=14)
ax.set_title('Relationship between initial velocity and launch angle for different
air densities', fontsize=16)
ax.grid(True, alpha=0.3)
ax.legend(fontsize=12)  # 显示图例区分不同 rho
```

```
plt.tight_layout()
plt.show()
```

Appendices 15

Introduction：Relationship between Angle and Initial Velocity (Showing Maximum Height)

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import warnings
warnings.filterwarnings('ignore')

# --------------------------- 1. 核心参数定义 ---------------------------
l = 2                # 初始高度相关参数 (m)，根据实际场景调整
g = 9.8                # 重力加速度 (m/s²)
m = 5                # 物体质量 (kg)，根据实际场景调整
r = 0.055                # 物体半径 (m)，根据实际场景调整
rho = 1.175            # 空气密度 (kg/m³)
C_d = 0.47             # 阻力系数 (球体默认 0.47)
u_x = 5.0              # x 方向风速 (m/s)，可调整
u_z = 2.0              # z 方向风速 (m/s)，可调整
k = 0.5 * C_d * rho * np.pi * r**2  # 阻力系数

# --------------------------- 2. 运动微分方程 ---------------------------
def motion(t, state):
    x, y, z, vx, vy, vz = state
    # 相对风速
    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    # 加速度计算
    ax = - (k / m) * v_rel * vrelx if v_rel > 1e-6 else 0.0
    ay = -g - (k / m) * v_rel * vrely if v_rel > 1e-6 else -g
    az = - (k / m) * v_rel * vrelz if v_rel > 1e-6 else 0.0
    return [vx, vy, vz, ax, ay, az]
```

```
# -------------------------- 3. 落地判定函数 --------------------------
def hit_ground(t, state):
    return state[1]
hit_ground.terminal = True
hit_ground.direction = -1

# 新增：最大高度判定函数（y 方向速度为 0 时）
def max_height(t, state):
    return state[4]  # vy=0 时达到最大高度
max_height.terminal = False  # 不终止积分
max_height.direction = -1    # 只检测 vy 从正变负的时刻

# -------------------------- 4. 目标函数与 v0 求解 --------------------------
def objective(v0, theta):
    x0 = 0.0
    y0 = 1 * np.sin(theta)
    z0 = 0.0
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    v0z = 0.0
    initial_state = [x0, y0, z0, v0x, v0y, v0z]
    t_span = [0, 10000]
    sol = solve_ivp(
        fun=motion, t_span=t_span, y0=initial_state,
        events=hit_ground, method='RK45', rtol=1e-6, atol=1e-9
    )
    if len(sol.t_events[0]) == 0:
        return 1e10
    x_landing = sol.y_events[0][0, 0]
    return x_landing - 1200

def find_v0(theta, v0_min=1.0, v0_max=1000.0, tol=1e-3, max_iter=100):
    f_min = objective(v0_min, theta)
    f_max = objective(v0_max, theta)
    # 调整搜索范围
    if f_min * f_max > 0:
```

```
        while f_min * f_max > 0 and v0_max < 1e5:
            v0_max *= 2
            f_max = objective(v0_max, theta)
        if f_min * f_max > 0:
            return None, None  # 同时返回 v0 和最大高度 h（无解时均为 None）
    # 二分迭代
    for _ in range(max_iter):
        v0_mid = (v0_min + v0_max) / 2
        f_mid = objective(v0_mid, theta)
        if abs(f_mid) < tol:
            # 计算对应最大高度 h
            initial_state = [0.0, l*np.sin(theta), 0.0, v0_mid*np.cos(theta),
v0_mid*np.sin(theta), 0.0]
            # 同时检测落地和最大高度两个事件
            sol = solve_ivp(fun=motion, t_span=[0,100], y0=initial_state,
                            events=[hit_ground, max_height], method='RK45')
            # 最大高度是 y 方向的最大值
            max_h = max(sol.y[1])  # 直接从求解结果中找 y 方向的最大值
            return v0_mid, max_h
        if f_min * f_mid < 0:
            v0_max = v0_mid
            f_max = f_mid
        else:
            v0_min = v0_mid
            f_min = f_mid
    # 迭代结束后计算最大高度 h
    initial_state = [0.0, l*np.sin(theta), 0.0, ((v0_min+v0_max)/2)*np.cos(theta),
((v0_min+v0_max)/2)*np.sin(theta), 0.0]
    sol = solve_ivp(fun=motion, t_span=[0,100], y0=initial_state,
                    events=[hit_ground, max_height], method='RK45')
    max_h = max(sol.y[1])  # 直接从求解结果中找 y 方向的最大值
    return (v0_min + v0_max) / 2, max_h

# -------------------------- 5. 遍历 theta 求解+结果可视化 ------------------------
--
theta_list = np.linspace(0, 80 * np.pi / 180, 30)  # 减少点数避免标注拥挤（30 个
```

```
点）
theta_deg = theta_list * 180 / np.pi
v0_list = []
max_h_list = []  # 存储最大高度

# 求解 v0 和对应的最大高度 h
for theta in theta_list:
    v0, max_h = find_v0(theta)
    if v0 is not None and max_h is not None:
        v0_list.append(v0)
        max_h_list.append(max_h)
    else:
        v0_list.append(np.nan)
        max_h_list.append(np.nan)

# --------------------------- 6. 绘制 v0-theta 图（带最大高度 h 标注） ---------------
------------
fig, ax = plt.subplots(1, 1, figsize=(12, 8))

# 绘制 v0-theta 曲线
ax.plot(theta_deg, v0_list, 'b-o', linewidth=2, markersize=6, label='v0-theta
relationship')

# 为每个有效点添加最大高度 h 标注
for i in range(len(theta_deg)):
    if not np.isnan(v0_list[i]) and not np.isnan(max_h_list[i]):
        # 标注格式：h=xx.xx m，位置偏移避免重叠
        ax.annotate(
            f'h={max_h_list[i]:.2f}m',
            xy=(theta_deg[i], v0_list[i]),
            xytext=(5, 5),  # 文字相对于点的偏移（x+5, y+5 像素）
            textcoords='offset points',
            fontsize=8,
            color='darkred',
            bbox=dict(boxstyle='round,pad=0.2', facecolor='white', alpha=0.7)   #
白色背景提高可读性
```

```
        )

# 图表美化
ax.set_xlabel('theta θ (° )', fontsize=14)
ax.set_ylabel('initial velocity v₀ (m/s)', fontsize=14)
ax.set_title('The relationship between initial velocity and launch angle (noting the maximum height h)', fontsize=16)
ax.grid(True, alpha=0.3)
ax.legend(fontsize=12)

plt.tight_layout()
plt.show()
```

Appendices 16

Introductiח： Images and Data of Initial Velocity vs. Angle at Different Heights

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
from scipy.optimize import fsolve

# -------------------------- 1. 统一定义已知参数 --------------------------
l = 2
g = 9.8
m = 5.0
r = 0.055
ρ = 1.175
C_d = 0.5
C_M = 0.2
u_x = 2.0
u_z = 1.0
ω0 = 50.0
A = np.pi * r**2
k = 0.5 * C_d * ρ * A

# 定义 7 个目标高度值（包含正负高度）
h_values = [0, 50, -50, 100, -100, -200, 200]  # 新增负高度：-50、-100、-200
```

```
# 原 5 个正向高度参考（如需单独运行可替换上方 h_values）：[0, 50, 100, 150, 200]

# -------------------------- 2. 运动微分方程 --------------------------
def motion_ode(t, state, v0, theta):
    x, y, z, vx, vy, vz = state
    omega_x = ω0 * np.cos(theta)
    omega_y = ω0 * np.sin(theta)

    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    v_rel = max(v_rel, 1e-6)  # 防止除以零

    F_Mx = 0.5 * C_M * ρ * A * r * (omega_y * vrelz)
    F_My = -0.5 * C_M * ρ * A * r * (omega_x * vrelz)
    F_Mz = 0.5 * C_M * ρ * A * r * (omega_x * vrely - omega_y * vrelx)

    ax = - (k/m) * v_rel * vrelx + F_Mx / m
    ay = -g - (k/m) * v_rel * vrely + F_My / m
    az = - (k/m) * v_rel * vrelz + F_Mz / m

    return [vx, vy, vz, ax, ay, az]

# -------------------------- 3. 到达目标位置判定函数 --------------------------
def target_condition(t, state, *args):
    x = state[0]
    return x - 1200.0  # 目标 x 坐标固定为 1200

target_condition.terminal = True  # 触发事件后终止积分
target_condition.direction = 1    # 仅当 x 从小于 1200 变为大于 1200 时触发

# -------------------------- 4. 目标函数 --------------------------
def target_function(v0, theta, target_y):
    v0 = float(v0)  # 确保 v0 为标量
    y0 = 1 * np.sin(theta)
```

```
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    initial_state = [0.0, y0, 0.0, v0x, v0y, 0.0]

    t_span = [0.0, 200.0]  # 足够长的时间跨度确保到达目标 x

    try:
        sol = solve_ivp(
            fun=motion_ode,
            t_span=t_span,
            y0=initial_state,
            args=(v0, theta),
            events=target_condition,
            method='RK45',
            rtol=1e-6,
            atol=1e-6
        )
    except Exception as e:
        print(f"求解器出错: v0={v0:.2f}, theta={np.degrees(theta):.2f}°, 错误:
{e}")
        return 1e10

    # 检查是否触发事件且获取有效 y 坐标
    if sol.status == 1 and len(sol.y_events[0]) > 0:
        y_final = sol.y_events[0][0][1]
        return (y_final - target_y)**2  # 返回 y 方向误差平方
    else:
        return 1e10  # 未到达目标 x，返回大误差

# --------------------------- 5. 求解初始速度 ---------------------------
def solve_v0_for_theta_h(theta, h):
    def error_func(v0):
        return np.sqrt(target_function(v0, theta, h))  # 误差开方，便于求解

    # 针对正负高度优化初始猜测（负高度所需速度更小）
    if h < 0:
```

```
        initial_guess = 150.0 + h * 0.3  # 负高度降低初始猜测
    else:
        initial_guess = 200.0 + h * 0.5  # 正高度按原逻辑猜测

    # 求解并转换为标量
    v0_solution, = fsolve(error_func, initial_guess)
    v0_solution = float(v0_solution)

    # 筛选合理解（速度在10-1000 m/s，误差小于1）
    if 10.0 < v0_solution < 1000.0:
        final_error = error_func(v0_solution)
        if final_error < 1.0:
            return v0_solution

    return None

# --------------------------- 6. 主程序（绘图+输出结果） ---------------------------
if __name__ == "__main__":
    theta_list_rad = np.linspace(0.1, 1.369, 25)  # 发射角度：20-70度（合理弹道范围）
    target_x = 1200

    # 绘图设置
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    fig, ax = plt.subplots(figsize=(14, 8))  # 扩大图表以容纳7条曲线图例

    # 遍历每个高度计算并绘图
    for h in h_values:
        v0_results = []
        valid_thetas_rad = []

        print(f"\n正在计算 h = {h} m 的情况...")
        for theta in theta_list_rad:
            v0 = solve_v0_for_theta_h(theta, h)
            if v0 is not None:
```

```
                valid_thetas_rad.append(theta)
                v0_results.append(v0)
                theta_deg = float(np.degrees(theta))
                print(f"   angle: {theta_deg:.2f}°, initial velocity: {v0:.2f}
m/s")

        # 绘制当前高度的曲线
        if valid_thetas_rad:
            valid_thetas_deg = np.degrees(valid_thetas_rad)
            ax.plot(valid_thetas_deg, v0_results, 'o-', label=f'h = {h} m',
markersize=4, linewidth=2)

    # 图表美化
    ax.set_title('The relationship between initial velocity and launch angle for
different target heights (at x = 1200m)', fontsize=16)
    ax.set_xlabel('launch angle (° )', fontsize=14)
    ax.set_ylabel('initial velocity (m/s)', fontsize=14)
    ax.grid(True, linestyle='--', alpha=0.6)
    ax.legend(fontsize=11, loc='upper center')  # 图例放在右上角避免遮挡
    ax.set_ylim(bottom=0)  # y 轴从 0 开始，符合速度物理意义

    plt.tight_layout()
    plt.show()
```

Appendices 17

Introductiո： Relationship between Angle and Initial Velocity (Including Maximum Height)

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# --------------------------- 1. 统一定义已知参数 ---------------------------
l = 2
g = 9.8
m = 5.0
r = 0.055
ρ = 1.175
```

```
C_d = 0.5
C_M = 0.2
u_x = 2.0
u_z = 1.0
ω0 = 50.0
A = np.pi * r**2
k = 0.5 * C_d * ρ * A

# --------------------------- 2. 运动微分方程 ---------------------------
def motion_ode(t, state, v0, theta):
    x, y, z, vx, vy, vz = state
    omega_x = ω0 * np.cos(theta)
    omega_y = ω0 * np.sin(theta)

    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    v_rel = max(v_rel, 1e-6)

    F_Mx = 0.5 * C_M * ρ * A * r * (omega_y * vrelz)
    F_My = -0.5 * C_M * ρ * A * r * (omega_x * vrelz)
    F_Mz = 0.5 * C_M * ρ * A * r * (omega_x * vrely - omega_y * vrelx)

    ax = - (k/m) * v_rel * vrelx + F_Mx / m
    ay = -g - (k/m) * v_rel * vrely + F_My / m
    az = - (k/m) * v_rel * vrelz + F_Mz / m

    return [vx, vy, vz, ax, ay, az]

# --------------------------- 3. 落地判定函数 ---------------------------
def landing_condition(t, state, v0, theta):
    return state[1]
landing_condition.terminal = True
landing_condition.direction = -1
```

```
# 新增：最大高度判定函数（y 方向速度为 0 时）
def max_height_condition(t, state, v0, theta):
    return state[4]  # vy=0 时触发
max_height_condition.terminal = False  # 不终止积分
max_height_condition.direction = 0     # 检测所有穿越零点的情况

# ---------------------------- 4. 目标函数（修改：同时返回 x 误差和最大高度 h）------
--------------------
def target_function(v0, theta):
    y0 = l * np.sin(theta)
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    initial_state = [0.0, y0, 0.0, v0x, v0y, 0.0]

    t_span = [0.0, 100.0]
    sol = solve_ivp(
        fun=motion_ode,
        t_span=t_span,
        y0=initial_state,
        args=(v0, theta),
        events=[landing_condition, max_height_condition],
        method='RK45',
        rtol=1e-6,
        atol=1e-6
    )

    x_landing = sol.y[0, -1]

    # 计算最大高度 h
    if len(sol.t_events[1]) > 0:  # 如果检测到了最大高度点
        max_height_time = sol.t_events[1][0]  # 取第一个最高点时间
        # 插值计算该时刻的 y 坐标（最大高度）
        max_height = np.interp(max_height_time, sol.t, sol.y[1])
    else:
        max_height = max(sol.y[1])  #  fallback：取轨迹中的最大 y 值
```

```
    return x_landing - 1200.0, max_height  # 返回 x 误差和最大高度

# --------------------------- 5. 二分法求解（修改：返回 v0 和对应的最大高度 h）-----
----------------------
def solve_v0_for_theta(theta, v0_min=10.0, v0_max=1000.0, tol=1e-3):
    f_min, _ = target_function(v0_min, theta)
    f_max, _ = target_function(v0_max, theta)

    if np.sign(f_min) == np.sign(f_max):
        return None, None  # 无解时返回两个 None

    iter_num = 0
    best_h = None
    while (v0_max - v0_min) > tol and iter_num < 100:
        v0_mid = (v0_min + v0_max) / 2
        f_mid, h_mid = target_function(v0_mid, theta)

        if np.abs(f_mid) < tol:
            best_h = h_mid
            break
        if np.sign(f_mid) == np.sign(f_min):
            v0_min = v0_mid
            f_min = f_mid
        else:
            v0_max = v0_mid
            f_max = f_mid
        iter_num += 1

    # 如果循环正常结束，再计算一次最终 v0 对应的最大高度
    if best_h is None:
        _, best_h = target_function(v0_mid, theta)

    return v0_mid, best_h

# --------------------------- 6. 主程序（修改：存储并绘制最大高度 h）--------------
------------
```

```
if __name__ == "__main__":
    theta_list_rad = np.linspace(0, 1.396, 30)
    v0_results = []
    h_results = []  # 存储最大高度 h
    valid_thetas_rad = []

    print("正在计算...")
    for theta in theta_list_rad:
        v0, max_h = solve_v0_for_theta(theta)
        if v0 is not None:
            valid_thetas_rad.append(theta)
            v0_results.append(v0)
            h_results.append(max_h)  # 存储最大高度

    # --- 绘图部分 ---
    if valid_thetas_rad:
        plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.rcParams['axes.unicode_minus'] = False

        fig, ax = plt.subplots(figsize=(12, 7))

        valid_thetas_deg = np.degrees(valid_thetas_rad)

        # 绘制 v0-theta 曲线和数据点
        ax.plot(valid_thetas_deg,     v0_results,     'b-o',     label='$v_0     -
\\theta$ relationship curve', markersize=6, linewidth=2)

        # 为每个数据点添加最大高度 h 标注
        for i, txt in enumerate(h_results):
            ax.annotate(f'h={txt:.2f}m',
                        xy=(valid_thetas_deg[i], v0_results[i]),
                        xytext=(5, 5),  # 文字相对于点的偏移量
                        textcoords='offset points',
                        fontsize=9,
                        color='darkred',
                        arrowprops=dict(arrowstyle='->', color='gray', lw=0.5))
```

```
        ax.set_title('The relationship between initial velocity and launch angle
(noting the maximum height h)', fontsize=16)
        ax.set_xlabel('launch angle $\\theta$ (° )', fontsize=14)
        ax.set_ylabel('initial velocity $v_0$ (m/s)', fontsize=14)

        ax.grid(True, linestyle='--', alpha=0.6)
        ax.legend(fontsize=12)

        # 调整 y 轴范围，给顶部的标注留出空间
        y_margin = (max(v0_results) - min(v0_results)) * 0.1
        ax.set_ylim(min(v0_results) - y_margin, max(v0_results) + y_margin * 2)

        plt.tight_layout()
        plt.show()
    else:
        print("在给定的角度范围内没有找到有效的解。")
```

Appendices 18

Introduction：Relationship between Angle and Initial Velocity (Including Final Kinetic Energy and Motion Time)

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# --------------------------- 1. 统一定义已知参数 ---------------------------
l = 2
g = 9.8
m = 5.0
r = 0.055
ρ = 1.175
C_d = 0.5
C_M = 0.2
u_x = 2.0
u_z = 1.0
ω0 = 50.0
A = np.pi * r**2
```

```
k = 0.5 * C_d * ρ * A

# --------------------------- 2. 运动微分方程 ---------------------------
def motion_ode(t, state, v0, theta):
    x, y, z, vx, vy, vz = state
    omega_x = ω0 * np.cos(theta)
    omega_y = ω0 * np.sin(theta)

    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    v_rel = max(v_rel, 1e-6)

    F_Mx = 0.5 * C_M * ρ * A * r * (omega_y * vrelz)
    F_My = -0.5 * C_M * ρ * A * r * (omega_x * vrelz)
    F_Mz = 0.5 * C_M * ρ * A * r * (omega_x * vrely - omega_y * vrelx)

    ax = - (k/m) * v_rel * vrelx + F_Mx / m
    ay = -g - (k/m) * v_rel * vrely + F_My / m
    az = - (k/m) * v_rel * vrelz + F_Mz / m

    return [vx, vy, vz, ax, ay, az]

# --------------------------- 3. 落地判定函数 ---------------------------
def landing_condition(t, state, v0, theta):
    return state[1]
landing_condition.terminal = True
landing_condition.direction = -1

# --------------------------- 4. 目标函数（修改：同时返回 x 误差、z 坐标、落地时间和末速度）---------------------------
def target_function(v0, theta):
    y0 = l * np.sin(theta)
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
```

```
    initial_state = [0.0, y0, 0.0, v0x, v0y, 0.0]

    t_span = [0.0, 100.0]
    sol = solve_ivp(
        fun=motion_ode,
        t_span=t_span,
        y0=initial_state,
        args=(v0, theta),
        events=landing_condition,
        method='RK45',
        rtol=1e-6,
        atol=1e-6
    )

    x_landing = sol.y[0, -1]
    z_landing = sol.y[2, -1]
    t_landing = sol.t[-1]  # 落地时间
    vx_final = sol.y[3, -1]  # 落地时 x 方向速度
    vy_final = sol.y[4, -1]  # 落地时 y 方向速度
    vz_final = sol.y[5, -1]  # 落地时 z 方向速度
    v_final = np.sqrt(vx_final**2 + vy_final**2 + vz_final**2)  # 合速度
    ke_final = 0.5 * m * v_final**2  # 最终动能

    return x_landing - 1200.0, z_landing, t_landing, ke_final

# -------------------------- 5. 二分法求解（修改：返回 v0、z 值、运动时间和最终动
能）--------------------------
def solve_v0_for_theta(theta, v0_min=10.0, v0_max=1000.0, tol=1e-3):
    f_min, _, _, _ = target_function(v0_min, theta)
    f_max, _, _, _ = target_function(v0_max, theta)

    if np.sign(f_min) == np.sign(f_max):
        return None, None, None, None  # 无解时返回四个 None

    iter_num = 0
    best_z = None
```

```
    best_t = None
    best_ke = None
    while (v0_max - v0_min) > tol and iter_num < 100:
        v0_mid = (v0_min + v0_max) / 2
        f_mid, z_mid, t_mid, ke_mid = target_function(v0_mid, theta)

        if np.abs(f_mid) < tol:
            best_z = z_mid
            best_t = t_mid
            best_ke = ke_mid
            break
        if np.sign(f_mid) == np.sign(f_min):
            v0_min = v0_mid
            f_min = f_mid
        else:
            v0_max = v0_mid
            f_max = f_mid
        iter_num += 1

    # 如果循环正常结束，再计算一次最终 v0 对应的参数
    if best_z is None:
        _, best_z, best_t, best_ke = target_function(v0_mid, theta)

    return v0_mid, best_z, best_t, best_ke

# --------------------------- 6. 主程序（修改：双坐标轴显示初速度、最终动能和运动时
间) ---------------------------
if __name__ == "__main__":
    theta_list_rad = np.linspace(0, 1.396, 30)
    v0_results = []
    t_results = []  # 运动时间
    ke_results = []  # 最终动能
    valid_thetas_rad = []

    print("正在计算...")
    for theta in theta_list_rad:
```

```
        v0, _, t_landing, ke_landing = solve_v0_for_theta(theta)
        if v0 is not None:
            valid_thetas_rad.append(theta)
            v0_results.append(v0)
            t_results.append(t_landing)
            ke_results.append(ke_landing)

    # --- 绘图部分 ---
    if valid_thetas_rad:
        plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.rcParams['axes.unicode_minus'] = False

        fig, ax1 = plt.subplots(figsize=(12, 7))

        valid_thetas_deg = np.degrees(valid_thetas_rad)

        # 左侧 y 轴：初速度
        color = 'tab:blue'
        ax1.set_xlabel('launch angle $\\theta$ (° )', fontsize=14)
        ax1.set_ylabel('initial velocity $v_0$ (m/s)', color=color, fontsize=14)
        ax1.plot(valid_thetas_deg, v0_results, 'b-o', label='initial velocity',
markersize=6, linewidth=2)
        ax1.tick_params(axis='y', labelcolor=color)
        ax1.grid(True, linestyle='--', alpha=0.6)
        ax1.legend(loc='upper left', fontsize=12)

        # 创建右侧 y 轴
        ax2 = ax1.twinx()
        color1 = 'tab:red'
        color2 = 'tab:green'

        # 右侧 y 轴第一个数据：最终动能
        ax2.plot(valid_thetas_deg, ke_results, 'r-s', label='final kinetic
energy', markersize=6, linewidth=2)
        ax2.set_ylabel('final kinetic energy (J)', color=color1, fontsize=14)
        ax2.tick_params(axis='y', labelcolor=color1)
```

```
        # 创建第三个坐标轴用于显示运动时间
        ax3 = ax1.twinx()
        # 将第三个坐标轴的位置向右移动，避免与第二个坐标轴重叠
        ax3.spines['right'].set_position(('outward', 60))
        ax3.plot(valid_thetas_deg,  t_results,  'g-^',  label='motion  time',
markersize=6, linewidth=2)
        ax3.set_ylabel('motion time (s)', color=color2, fontsize=14)
        ax3.tick_params(axis='y', labelcolor=color2)

        # 合并图例
        lines1, labels1 = ax1.get_legend_handles_labels()
        lines2, labels2 = ax2.get_legend_handles_labels()
        lines3, labels3 = ax3.get_legend_handles_labels()
        ax1.legend(lines1  +  lines2  +  lines3,  labels1  +  labels2  +  labels3,
loc='upper center', fontsize=12)

        ax1.set_title('The relationship between initial velocity, final kinetic
energy, time of motion and launch angle', fontsize=16)

        plt.tight_layout()
        plt.show()
    else:
        print("在给定的角度范围内没有找到有效的解。")
```

Appendices 19

Introductiո： Relationship between Angle and Initial Velocity (Considering Wind Speed and Rotation)

```
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

# --------------------------- 1. 统一定义已知参数 ---------------------------
l = 2
g = 9.8
m = 5.0
r = 0.055
ρ = 1.175
C_d = 0.5
C_M = 0.2
u_x = 2.0
u_z = 1.0
ω0 = 50.0
A = np.pi * r**2
k = 0.5 * C_d * ρ * A

# --------------------------- 2. 运动微分方程 ---------------------------
def motion_ode(t, state, v0, theta):
    x, y, z, vx, vy, vz = state
    omega_x = ω0 * np.cos(theta)
    omega_y = ω0 * np.sin(theta)

    vrelx = vx - u_x
    vrely = vy
    vrelz = vz - u_z
    v_rel = np.sqrt(vrelx**2 + vrely**2 + vrelz**2)
    v_rel = max(v_rel, 1e-6)

    F_Mx = 0.5 * C_M * ρ * A * r * (omega_y * vrelz)
    F_My = -0.5 * C_M * ρ * A * r * (omega_x * vrelz)
    F_Mz = 0.5 * C_M * ρ * A * r * (omega_x * vrely - omega_y * vrelx)

    ax = - (k/m) * v_rel * vrelx + F_Mx / m
    ay = -g - (k/m) * v_rel * vrely + F_My / m
```

```
    az = - (k/m) * v_rel * vrelz + F_Mz / m

    return [vx, vy, vz, ax, ay, az]

# --------------------------- 3. 落地判定函数 ---------------------------
def landing_condition(t, state, v0, theta):
    return state[1]
landing_condition.terminal = True
landing_condition.direction = -1

# --------------------------- 4. 目标函数（修改：同时返回 x 误差和 z 坐标） ----------
----------------
def target_function(v0, theta):
    y0 = l * np.sin(theta)
    v0x = v0 * np.cos(theta)
    v0y = v0 * np.sin(theta)
    initial_state = [0.0, y0, 0.0, v0x, v0y, 0.0]

    t_span = [0.0, 100.0]
    sol = solve_ivp(
        fun=motion_ode,
        t_span=t_span,
        y0=initial_state,
        args=(v0, theta),
        events=landing_condition,
        method='RK45',
        rtol=1e-6,
        atol=1e-6
    )

    x_landing = sol.y[0, -1]
    z_landing = sol.y[2, -1] # 获取落地时的 z 坐标
    return x_landing - 1200.0, z_landing # 返回 x 误差和 z 坐标

# --------------------------- 5. 二分法求解（修改：返回 v0 和对应的 z 值） ----------
---------------
```

```
def solve_v0_for_theta(theta, v0_min=10.0, v0_max=1000.0, tol=1e-3):
    f_min, _ = target_function(v0_min, theta)
    f_max, _ = target_function(v0_max, theta)

    if np.sign(f_min) == np.sign(f_max):
        return None, None # 无解时返回两个 None

    iter_num = 0
    best_z = None
    while (v0_max - v0_min) > tol and iter_num < 100:
        v0_mid = (v0_min + v0_max) / 2
        f_mid, z_mid = target_function(v0_mid, theta)

        if np.abs(f_mid) < tol:
            best_z = z_mid
            break
        if np.sign(f_mid) == np.sign(f_min):
            v0_min = v0_mid
            f_min = f_mid
        else:
            v0_max = v0_mid
            f_max = f_mid
        iter_num += 1

    # 如果循环正常结束，再计算一次最终 v0 对应的 z 值
    if best_z is None:
        _, best_z = target_function(v0_mid, theta)

    return v0_mid, best_z

# ---------------------------- 6. 主程序（修改：存储并绘制 z 值）----------------------
------
if __name__ == "__main__":
    theta_list_rad = np.linspace(0, 1.396, 30)
    v0_results = []
    z_results = [] # 新增：存储落地时的 z 坐标
```

```
    valid_thetas_rad = []

    print("正在计算...")
    for theta in theta_list_rad:
        v0, z_landing = solve_v0_for_theta(theta)
        if v0 is not None:
            valid_thetas_rad.append(theta)
            v0_results.append(v0)
            z_results.append(z_landing) # 存储 z 值

    # --- 绘图部分 ---
    if valid_thetas_rad:
        plt.rcParams['font.sans-serif'] = ['SimHei']
        plt.rcParams['axes.unicode_minus'] = False

        fig, ax = plt.subplots(figsize=(12, 7)) # 稍微调大图表以容纳文字

        valid_thetas_deg = np.degrees(valid_thetas_rad)

        # 绘制 v0-theta 曲线和数据点
        ax.plot(valid_thetas_deg,     v0_results,     'b-o',     label='$v_0     -
\\theta$ relationship curve', markersize=6, linewidth=2)

        # 为每个数据点添加 z 值标注
        for i, txt in enumerate(z_results):
            # ax.text(x, y, 文本, ...)
            # xytext 和 arrowprops 用于添加一个指向点的箭头，避免文字覆盖
            ax.annotate(f'z={txt:.2f}m',
                        xy=(valid_thetas_deg[i], v0_results[i]),
                        xytext=(5, 5), # 文字相对于点的偏移量
                        textcoords='offset points',
                        fontsize=9,
                        color='darkred',
                        arrowprops=dict(arrowstyle='->', color='gray', lw=0.5))

        ax.set_title('The relationship between initial velocity and launch angle
```

```
(noting the Z-value at landing)', fontsize=16)
        ax.set_xlabel('launch angle  $\\theta$ (° )', fontsize=14)
        ax.set_ylabel('initial velocity $v_0$ (m/s)', fontsize=14)

        ax.grid(True, linestyle='--', alpha=0.6)
        ax.legend(fontsize=12)

        # 调整 y 轴范围，给顶部的标注留出空间
        y_margin = (max(v0_results) - min(v0_results)) * 0.1
        ax.set_ylim(min(v0_results) - y_margin, max(v0_results) + y_margin * 2)

        plt.tight_layout()
        plt.show()
    else:
        print("在给定的角度范围内没有找到有效的解。")
```

序号	x (m)	y (m)	uwx (m/s)	uz (m/s)	theta (°)
1	1142.86	450.77	5.0	10.0	25.82
2	1142.86	372.73	0.0	6.0	22.14
3	1500.00	643.54	5.0	4.0	30.71
4	1071.43	54.27	10.0	6.0	6.22
5	1357.14	617.76	7.5	12.0	30.71
6	1428.57	406.59	10.0	6.0	22.14
7	1214.29	1469.10	10.0	6.0	60.10
8	1000.00	273.07	5.0	4.0	18.47
9	1071.43	167.28	10.0	10.0	12.35
10	1142.86	595.37	0.0	10.0	31.94
11	1428.57	1370.36	7.5	12.0	55.20
12	1000.00	1502.27	7.5	6.0	63.78
13	1142.86	504.28	7.5	10.0	28.27
14	1357.14	371.70	2.5	6.0	20.92
15	1500.00	678.11	5.0	8.0	31.94
16	1357.14	341.92	2.5	10.0	19.69
17	1357.14	521.70	10.0	6.0	27.04
18	1000.00	141.78	5.0	10.0	11.12
19	1000.00	206.09	10.0	6.0	14.80
20	1357.14	283.10	2.5	4.0	17.24
21	1500.00	1408.41	0.0	6.0	62.55
22	1071.43	647.99	10.0	10.0	35.61
23	1000.00	120.42	10.0	8.0	9.90
24	1500.00	1234.98	0.0	8.0	50.31

25	1142.86	535.44	2.5	12.0	29.49
26	1285.71	670.46	0.0	12.0	33.16
27	1000.00	951.55	0.0	4.0	47.86
28	1142.86	880.50	5.0	4.0	42.96
29	1428.57	1257.80	7.5	8.0	51.53
30	1285.71	668.51	2.5	8.0	33.16
31	1214.29	802.15	10.0	6.0	39.29
32	1142.86	1488.41	7.5	8.0	61.33
33	1285.71	699.14	5.0	4.0	34.39
34	1071.43	878.05	5.0	6.0	44.18
35	1500.00	966.16	7.5	12.0	41.73
36	1071.43	283.87	7.5	4.0	18.47
37	1142.86	532.25	7.5	10.0	29.49
38	1428.57	44.22	5.0	12.0	7.45
39	1500.00	1161.17	0.0	10.0	47.86
40	1071.43	1476.09	0.0	4.0	61.33
41	1071.43	1262.96	10.0	6.0	56.43
42	1285.71	394.48	0.0	10.0	22.14
43	1000.00	1289.34	0.0	6.0	57.65
44	1500.00	251.79	7.5	6.0	16.02
45	1000.00	1310.94	7.5	10.0	58.88
46	1428.57	375.96	0.0	4.0	20.92
47	1000.00	228.24	7.5	6.0	16.02
48	1357.14	1194.80	10.0	4.0	50.31
49	1071.43	1322.72	5.0	8.0	57.65
50	1000.00	1133.02	7.5	10.0	53.98