DIGITAL MATHEMATICS APPLIED IN DEFENCE AND SECURITY EDUCATION (DIMAS)

KA220-HED - Cooperation Partnerships in Higher Education

2023-1-BG01-KA220-HED-000156664

SCENARIO TEMPLATE

Version 1.0

October 2025

SCENARIO 1.2

Title	**Modelling Projectile Trajectory with Aerodynamic Drag**
Short description	This scenario provides military engineering students with a practical simulation of projectile motion under realistic battlefield conditions. It models the trajectory of a projectile while accounting for aerodynamic drag, gravity, and launch parameters such as muzzle velocity and firing angle. The exercise helps students understand the influence of aerodynamic forces on range, accuracy, and impact point—key factors in the design, testing, and operational use of modern weapon systems.
Topics Involved	- Fundamentals of projectile motion - Aerodynamic drag and drag coefficient - Ballistic equations of motion - Numerical modelling and simulation techniques - Effects of launch parameters (velocity, angle, mass) - External forces: gravity and air resistance - Computational tools for trajectory prediction - Application of physics in weapon system design - Data analysis and visualization of projectile paths - Accuracy and range optimization in military ballistics
Areas of the mathematics	- **Differential Equations** – to describe the projectile's velocity and position over time - **Numerical Analysis** – for solving nonlinear equations that lack analytical solutions - **Vector Calculus** – to represent motion and forces in two or three dimensions - **Algebra and Trigonometry** – for resolving components of velocity and angle relationships - **Applied Mathematics** – integrating physical models with computational simulation - **Statistics and Data Analysis** – for evaluating trajectory data and error estimation
Digital mathematics tools	Matlab
Learning objectives (knowledge, abilities, competencies)	Knowledge: - Understand the physical principles governing projectile motion with and without aerodynamic drag.

	- Explain how drag coefficients, air density, and launch parameters influence trajectory behaviour. - Identify the mathematical models used to describe projectile dynamics in military applications. Abilities: - Apply differential equations to simulate projectile trajectories under real-world conditions. - Use computational tools to model and visualize projectile flight paths. - Analyze and interpret simulation results to assess range, accuracy, and impact energy. Competencies: - Develop the capability to integrate mathematical modeling with engineering judgment in ballistic analysis. - Demonstrate critical thinking in optimizing projectile performance considering aerodynamic effects. - Collaborate effectively in technical problem-solving related to weapon system design and testing.
Methodologies adopted	1. Analytical modelling - Derive closed-form relationships where possible (ideal projectile without drag; small-angle approximations). Use these solutions to build intuition, set initial conditions, and provide benchmarks for numerical results. 2. Derivation of governing equations- Formulate the full equations of motion including gravity, quadratic (or other) aerodynamic drag, and optionally lift and Coriolis force. Express in vector form and split into components for 2-D or 3-D analysis. 3. Dimensional analysis - Introduce characteristic scales (length, time, velocity) and no dimensional numbers (e.g., ballistic coefficient, Reynolds number) to identify dominating effects and simplify parameter studies. 4. Numerical integration of ODEs - Solve the nonlinear equations of motion using reliable time-stepping ODE solvers (e.g., Runge–Kutta 4(5), adaptive step solvers). Implement event detection (impact, max altitude) and ensure solver stability and error control. 5. Aerodynamic modelling Use empirical drag laws (constant C_d, speed-dependent $C_d(V)$, piecewise regimes) and tabulated ballistic coefficient data. 6. Numerical methods - Apply step-size control, stiffness detection, and convergence testing. Perform grid/sensitivity refinement and compare multiple integrators to quantify numerical error.

	7. Parameter estimation and calibration - Fit drag coefficients or ballistic coefficients to experimental or published trajectory data using least-squares or Bayesian estimation to improve model fidelity. 8. Sensitivity and uncertainty analysis - Perform one-factor-at-a-time and global sensitivity analyses (Sobol, Morris) and Monte Carlo simulations to quantify how uncertainties in initial conditions, mass, drag, or atmospheric parameters affect range and impact point. 9. Environmental modelling - Include air density variation with altitude, wind profiles, temperature, and humidity. Model Coriolis and geodetic effects for long-range trajectories when relevant. 10. Validation and verification - Verify numerical code against analytical solutions and published ballistic tables. Validate models against experimental firing data or trusted simulations. Document test cases and discrepancies. 11. Optimization studies - Use optimization algorithms (gradient-based or heuristic) to find launch parameters (angle, velocity) or design parameters (mass, shape) that maximize range, minimize dispersion, or meet mission constraints. 12. Visualization and data analysis -Plot trajectories, phase-space diagrams, range vs. angle curves, and error bands. Use statistical summaries and reporting to interpret results and support engineering decisions. 13. Software engineering practices - Adopt version control, modular code structure, unit tests for physics routines, and reproducible notebooks/scripts to ensure transparency and repeatability.
Prerequisites	Mathematics & numerical skills - Numerical methods for ODEs (Runge–Kutta family, stability concepts). - Basic error analysis and numerical convergence testing. - Familiarity with statistics (mean, variance, basic Monte Carlo concepts) for uncertainty analysis. Programming & software - Proficient in a scientific scripting language (MATLAB). - Ability to implement numerical integrators, read/write data files, and produce plots.
Estimated time	8 hours (including self-studies and teamwork)

Task for students	**Mission Brief** As part of your military engineering training, you are assigned to simulate the trajectory of a projectile under realistic battlefield conditions. The mission involves developing a computational model that accounts for aerodynamic drag, gravity, and launch parameters to predict the projectile's flight path, range, and impact characteristics. Your analysis will support decision-making in weapon system design, targeting accuracy, and operational planning. • Define initial conditions and projectile parameters • Formulate the system of ordinary differential equations (ODEs) • Perform numerical integration and detect the impact point • Extract trajectory features (maximum height, range) • Visualize the results Task 1. For each function write a one–paragraph summary ("What it does, inputs & outputs, key steps"). Task 2. Trace the data flow: draw a simple block diagram showing how data moves from *initial_coditions* → _____ → *trajectory_animation*. Task 3. Write a script called *parameter_variation* in which you: - Vary one parameter at a time (Launch angle θ_0, mass m, or frontal area S; Keep the other two parameters constant). - For each variation: compute the range and the apex (maximum height) and plot how each output (range, height) depends on the chosen parameter. - Write a short report in which interpret the observed trends (for example, how increasing mass alters the effect of drag).
Assessment	1. Theoretical Understanding (20%) - Demonstrates clear understanding of projectile motion principles, drag modelling, and governing equations. 2. Model Development & Implementation (30%) - Correct formulation and accurate numerical implementation (e.g., ODE solver setup, code efficiency, documentation.

	3. Simulation & Data Analysis (20%) - Quality and accuracy of simulation results; correctness of computed parameters (range, flight time, velocity, etc.); analysis of drag influence. 4. Results Interpretation & Discussion (20%) - Ability to interpret findings, assess operational significance (accuracy, range optimization), and propose improvements. 5. Report Quality (10%) - Structure, clarity, use of figures/tables, professional technical writing, and adherence to report format.

Solution

In projectile motion, aerodynamic drag fundamentally alters the trajectory compared to the ideal parabolic path. We consider a projectile of mass *m*, frontal area *S*, and drag coefficient *c_x*, moving under gravity *g* in an atmosphere where density varies with altitude *y*.

Governing Equations

Model Assumptions: No wind; motion confined to a vertical plane; constant gravitational acceleration g; neglect Coriolis and lift effects.

Let *V(t)* be the speed, *θ(t)* the trajectory angle above the horizontal, and *(x(t), y(t))* the coordinates. The system of ordinary differential equations is:

$$\begin{aligned}
\frac{dV}{dt} &= -\frac{1}{2m}\,\rho(y)\,S\,c_x\big(M\big)\,V^2 \;-\; g\sin\theta,\\
\frac{d\theta}{dt} &= -\frac{g\cos\theta}{V},\\
\frac{dx}{dt} &= V\cos\theta,\\
\frac{dy}{dt} &= V\sin\theta.
\end{aligned}$$

- V: Projectile speed (m/s)
- θ: Trajectory angle (radians)
- x, y: Horizontal and vertical positions (meters)
- M: Mach number = V/a(y), where a(y) is the local speed of sound

Atmospheric and Aerodynamic Models

- **Air Density (ρ(y))**

  Modeled by an exponential decay:

$$\rho(y) = \rho_0\ \exp\big(-y/H\big),$$

  where ρ_0 is sea-level density (~1.225 kg/m³) and H ≈ 8500 m is the scale height.

- Speed of Sound (a(y))

  Varies with temperature and altitude—often approximated by piecewise linear or polynomial fits based on the standard atmosphere.

- Drag Coefficient (c_x(M))

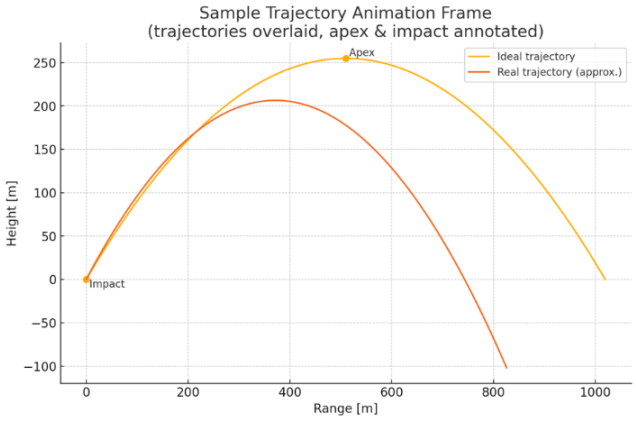
  Described by the Siacci law or tabulated data, c_x depends on Mach number

$$c_x = c_x(M),$$

Capturing subsonic, transonic, and supersonic regimes.

Key Insights

- The coupling between velocity V and angle θ demonstrates that drag not only reduces speed but also gradually lowers the trajectory angle compared to drag-free motion, leading to a flatter path.
- Unlike drag-free motion (where dθ/dt = 0), the gravitational term affects θ over time, resulting in an asymmetric ascent and descent.
- Numerical integration is required because the drag coefficient c_x varies nonlinearly with Mach number, and air density σ changes with altitude, preventing a closed-form solution.
- ***Concrete Example*:** For a 0.425 kg projectile launched at 45°, the real trajectory reaches a maximum height of approximately 14.6 km and a range of approximately 60 km, compared to about 16.3 km and 65 km for the ideal case.



1. Supporting functions:

- `ro(y)`: air density at altitude y (standard atmosphere model)
- `fsound(y)`: speed of sound at altitude y
- `cxsiacci(M)`: drag coefficient as a function of Mach number (Siacci law)

A simplified model assumes exponential density decay and empirical speed of sound:

- Density: where $H \approx 8500$ m is the scale height.
- Speed of sound: approximated by piecewise or empirical formulas.

2. MATLAB Script Structure

2.1. Directory Structure

- `init_cond.m` (initial conditions)

```
function [ci]=init_cond

% Function for defining the initial firing conditions
% INPUT:  -
% OUTPUT:
%     ci     - structure containing the initial firing conditions
%     ci.X0  - X coordinate of the firing position [m]
%     ci.Y0  - Y coordinate of the firing position [m]
%     ci.V0  - initial velocity relative to the air [m/s]
%     ci.teta0 - initial firing angle [rad]
% CALL:   ci = cond_init
% Firing position
X0=0;%[m]
Y0=3000;%[m]
% Initial velocity
V0=1070+20;%[m/s]
% Firing angle
teta0=0+5; %[grade]
% Grouping the initial conditions into a structure ci
ci.X0=X0;%[m]
ci.Y0=Y0;%[m]
ci.V0=V0;%[m/s]
ci.teta0=teta0*pi/180; %[rad]
```

- `projectile_parameters.m` (projectile parameters)

```
function par=projectile_parameters
% Function for defining the physical and numerical parameters of the
weapon-ammunition system
% necessary for trajectory calculation
% INPUT:  -
% OUTPUT:
%     par    - structure containing the weapon-ammunition system
parameters
%     par.d    - projectile caliber [m]
%     par.m    - projectile mass [kg]
%     par.q    - projectile weight [kgf]
%     par.ind  - Siacci form factor [-]
%     par.g    - gravitational acceleration at ground level [m/s^2]
%     par.t0   - initial time to start the trajectory [s]
%     par.tmax - final time until which the trajectory is calculated [s]
%     par.n    - number of points along the trajectory [-]
% CALL:
%     par = param_proiectil
% Projectile caliber
```

```
d=30; %[mm]
% Projectile mass
m=0.425;% [kg]
% Form factor for the Siacci drag law
ind=0.740;%[]
% Gravitational acceleration at ground level
g=9.80665;% [m/s^2]
% Initial time at which the trajectory starts [s]
t0=0;
% Final time up to which the trajectory is calculated [s]
tmax=100;
% Number of points along the trajectory []
n=100;
% Grouping the weapon-ammunition system parameters into a structure par
par.d=d/1000; %[m]
par.m=m; %[kg]
par.q=m*g; %[kgf]
par.ind=ind; %[]
par.g=g;%[m/s^2]
par.t0=t0;%[s]
par.tmax=tmax;%[s]
par.n=n; %[]
```

- `real_trajectory_model.m` (defines the ODE)

```
function Wp=real_trajectory_model(t,W,par)
% Function for defining the model of the system of differential
equations
% for the trajectory of a projectile in air
% INPUT:
%     t   - time [s]
%     W   - vector of the system's state variables
%           W(1) = V     - projectile velocity [m/s]
%           W(2) = theta - trajectory inclination [rad]
%           W(3) = x     - x coordinate [m]
%           W(4) = y     - y coordinate [m]
%     par - structure containing the weapon-ammunition system parameters
%           par.d    - projectile caliber [m]
%           par.m    - projectile mass [kg]
%           par.q    - projectile weight [kgf]
%           par.ind  - Siacci form factor [-]
%           par.g    - gravitational acceleration at ground level[m/s^2]
%           par.t0   - initial time to start the trajectory [s]
%           par.tmax - final time to end calculations [s]
%           par.n    - number of points along the trajectory [-]
% OUTPUT:
%     Wp - derivatives of the state variables (model
%           of the system of differential equations)
%           Wp(1) = dV/dt     - projectile acceleration [m/s^2]
%           Wp(2) = dtheta/dt - rate of change of trajectory inclination
[rad/s]
%           Wp(3) = dx/dt     - velocity along the x-axis [m/s]
```

```
%           Wp(4) = dy/dt     - velocity along the y-axis [m/s]
V=W(1); %m/s
teta=W(2); %rad
x=W(3); %m
y=W(4); %m
% Gravitational acceleration at ground level
g=par.g;
% velocity of sound:
a = fsound(y);
% Siacci form factor
ind=par.ind;
% Drag coefficient
cx=ind*cxsiacci(V/a);
% Projectile caliber
d=par.d;
% Projectile mass
m=par.m;
% Reference area of the projectile
S=pi*d^2/4;
Kx=ro(y)*S*cx/2/m;
Wp(1)=-Kx*V^2-g*sin(teta);
Wp(2)=-g*cos(teta)/V;
Wp(3)=V*cos(teta);
Wp(4)=V*sin(teta);
Wp=Wp';
function r = ro(y)
% ro   - air density as a function of altitude
% y - altitude at which the air density is determined [m]
% r = ro(y) - air density at altitude y [kg/m^3]
if y <= 11000
   r = 1.2255*(1 - y/44308)^4.2553;
else
   r = exp((11000 - y)/14600)*0.36385;
end
function cx=cxsiacci(M)
% Function for calculating Siacci drag coefficients
% INPUT:
%     M  - Mach number, M = V/a (vector or scalar)
% OUTPUT:
%     cx - Siacci drag coefficient
% CALL:
%     cx = cxsiacci(M)

% Tabulated values of Mach number and the corresponding Siacci drag
coefficient
% for Mach numbers in the range 0.1...4
tabel=       [0.1 0.255; 0.2 0.255; 0.3 0.256; 0.4 0.256; 0.5 0.257];
tabel=[tabel; 0.6 0.259; 0.7 0.266; 0.8 0.285; 0.9 0.405; 1.0 0.546];
tabel=[tabel; 1.1 0.639; 1.2 0.690; 1.3 0.718; 1.4 0.731; 1.5 0.734];
tabel=[tabel; 1.6 0.734; 1.7 0.728; 1.8 0.718; 1.9 0.707; 2.0 0.694];
tabel=[tabel; 2.1 0.681; 2.2 0.667; 2.3 0.653; 2.4 0.639; 2.5 0.625];
tabel=[tabel; 2.6 0.611; 2.7 0.597; 2.8 0.585; 2.9 0.572; 3.0 0.559];
```

```
tabel=[tabel; 3.1 0.546; 3.2 0.536; 3.3 0.524; 3.4 0.514; 3.5 0.503];
tabel=[tabel; 3.6 0.493; 3.7 0.483; 3.8 0.474; 3.9 0.464; 4.0 0.456];
% Identification of the tabulated values for Mach number
Mt=tabel(:,1);
% Identification of tabulated values for the Siacci drag coefficient
cxt=tabel(:,2);
% Identification of values for which the drag coefficient needs to be
calculated
% that are not covered by the table
i=find(M<0.1);
M(i)=0.1;
i=find(M>4);
M(i)=4;
% Calculation of the drag coefficient by interpolation
cx=interp1(Mt,cxt,M);
function a = fsound(y)
% Function for calculating the function a(y), which computes
% the speed of sound at the current point at altitude y
% for the International Standard Atmosphere
% INPUT:
%    y – altitude at which the speed of sound is determined [m]
% OUTPUT:
%    a – speed of sound at altitude y [m/s]
% CALL:
%    a = fsound(y)
% Air temperature at altitude y
temp=fT(y);%[K]
% Adiabatic coefficient
gamma=1.408;
% Universal gas constant
R=8314.51;%[J/(kmol*K)]
% Molar mass of dry air
M=28.9647;%[kg/kmol]
% Speed of sound
a=sqrt(gamma*R*temp/M);
function t = fT(y)
% Function for calculating T(y), which computes
% the air temperature at the current point at altitude y
% based on the air temperature at ground level (temp0)
% for the International Standard Atmosphere
% temp0 = 288; % K
% INPUT:
%    y – altitude at which the air temperature is determined [m]
% OUTPUT:
%    t – temperature at altitude y [K]
% CALL:
%    t = fT(y)
temp0=288;%
if y <= 11000
    t=temp0-0.0065*y;
else
    t=temp0-0.0065*11000;
```

```
    end
```

- `real_trajectory_num.m` (numerical integration)

```
    function [tr]=real_trajectory_num(ci,par)
    % Function for determining the main trajectory elements in air
    % using numerical integration
    % INPUT:
    %     ci   - structure containing the initial firing conditions
    %            ci.X0    - X coordinate of the firing position [m]
    %            ci.Y0    - Y coordinate of the firing position [m]
    %            ci.V0    - initial velocity relative to the air [m/s]
    %            ci.theta0 - initial firing angle [rad]
    %     par  - structure containing the weapon-ammunition system
    parameters
    %            par.d     - projectile caliber [m]
    %            par.m     - projectile mass [kg]
    %            par.q     - projectile weight [kgf]
    %            par.i     - Siacci form factor [-]
    %            par.g     - gravitational acceleration at ground level
    [m/s^2]
    %            par.t0    - initial time at which the trajectory starts [s]
    %            par.tmax  - final time up to which the trajectory is
    calculated [s]
    %            par.n     - number of points along the trajectory [-]
    % OUTPUT:
    %     tr   - structure containing the main trajectory elements
    %            tr.n     - number of trajectory points [-]
    %            tr.t     - trajectory duration - vector of n elements [s]
    %            tr.x     - x coordinate of trajectory points - vector of n
    elements [m]
    %            tr.y     - y coordinate of trajectory points - vector of n
    elements [m]
    %            tr.V     - projectile velocity - vector of n elements [m/s]
    %            tr.theta - trajectory inclination angle - vector of n
    elements [rad]
    % CALL:
    %     [tr] = traiect_real_num(ci, par)
    % Extraction of initial conditions
    X0=ci.X0;
    Y0=ci.Y0;
    V0=ci.V0;
    teta0=ci.teta0;
    W0=[V0; teta0; X0; Y0];
    % Extraction of the weapon-ammunition system parameters
    n=par.n;
    t0=par.t0;
    tmax=par.tmax;
    % Stop condition for ending the simulation upon impact with the ground
    ops = odeset('Events',@stop);
    % Integration of the system of equations using the Runge-Kutta method
```

```
[tc,Wc]=ode45(@(t,W) real_trajectory_model(t,W,par),[t0:0.01:tmax],
W0,ops);
Vc=Wc(:,1);
tetac=Wc(:,2);
xc=Wc(:,3);
yc=Wc(:,4);
%Interpolation of the results to obtain n points
timp=tc(length(tc));
dt=(timp-t0)/(n-1);
t=t0:dt:timp;t(n)=timp;t=t';
V=interp1(tc,Vc,t);
teta=interp1(tc,tetac,t);
x=interp1(tc,xc,t);
y=interp1(tc,yc,t);
% Structure containing the trajectory elements
tr.n=n;
tr.t=t;
tr.x=x;
tr.y=y;
tr.V=V;
tr.teta=teta;
function [value,isterminal,direction] = stop(t,y)
% Locate the time when height passes through zero in a decreasing
direction and stop integration
value = y(4);     % Detect height = 0
isterminal = 1;   % Stop the integration
direction = -1;   % Negative direction only
```

- `impact_point.m`, `apex.m`, `air_trajectory.m`, `animation_trajectory.m` (analysis and plotting)

```
function [imp]=impact_point(tr)
% Function for extracting the main trajectory elements at the impact
point
% INPUT:
%     tr - structure containing the main trajectory elements
%          tr.n     - number of trajectory points [-]
%          tr.t     - trajectory duration - vector of n elements [s]
%          tr.x     - x coordinate of trajectory points - vector of n
elements [m]
%          tr.y     - y coordinate of trajectory points - vector of n
elements [m]
%          tr.V     - projectile velocity - vector of n elements [m/s]
%          tr.theta - trajectory inclination angle - vector of n
elements [rad]
% OUTPUT:
%     imp - structure containing the main trajectory elements at the
impact point
%          imp.t     - trajectory duration until impact [s]
%          imp.x     - x coordinate of impact point [m]
%          imp.y     - y coordinate of impact point [m]
%          imp.V     - projectile velocity at impact [m/s]
```

```
    %          imp.theta - trajectory inclination angle at impact [rad]
    % CALL:
    %     [imp] = punct_impact(tr)
    % Extraction of the main trajectory elements
    n=tr.n;
    t=tr.t;
    x=tr.x;
    y=tr.y;
    V=tr.V;
    teta=tr.teta;
    % Identification of the impact point and interpolations
    yimp=0;
    timp=interp1(y(n/2:n),t(n/2:n),yimp)
    ximp=interp1(y(n/2:n),x(n/2:n),yimp);
    Vimp=interp1(y(n/2:n),V(n/2:n),yimp);
    tetaimp=interp1(y(n/2:n),teta(n/2:n),yimp);
    % Structure containing the trajectory elements at the impact point
    imp.t=timp;
    imp.x=ximp;
    imp.y=yimp;
    imp.V=Vimp;
    imp.teta=tetaimp;
```

```
function [sag]=apex(tr)
% Function for extracting the main trajectory elements at the apex
% INPUT:
%     tr - structure containing the main trajectory elements
%          tr.n     - number of trajectory points [-]
%          tr.t     - trajectory duration - vector of n elements [s]
%          tr.x     - x coordinate of trajectory points - vector of n
elements [m]
%          tr.y     - y coordinate of trajectory points - vector of n
elements [m]
%          tr.V     - projectile velocity - vector of n elements [m/s]
%          tr.theta - trajectory inclination angle - vector of n elements
[rad]
% OUTPUT:
%     sag - structure containing the main trajectory elements at the apex
%          sag.t     - time to apex [s]
%          sag.x     - x coordinate of the apex [m]
%          sag.y     - y coordinate of the apex [m]
%          sag.V     - projectile velocity at the apex [m/s]
%          sag.theta - trajectory inclination angle at the apex [rad]
% CALL:
%     [sag] = sageata(tr)
% Extraction of the main trajectory elements
n=tr.n;
t=tr.t;
x=tr.x;
y=tr.y;
V=tr.V;
teta=tr.teta;
% Identification of the trajectory apex and interpolations
```

```
tetasag=0;
tsag=interp1(teta,t,tetasag);
xsag=interp1(teta,x,tetasag);
ysag=interp1(teta,y,tetasag);
Vsag=interp1(teta,V,tetasag);
% Structure containing the trajectory elements at the impact point
sag.t=tsag;
sag.x=xsag;
sag.y=ysag;
sag.V=Vsag;
sag.teta=tetasag;
```

```
% compute the full ballistic trajectory under drag & gravity,
% Extract the key points (launch, apex, impact),
% produce a static 2D plot of y vs. x highlighting those three points.
% Initial conditions
ci=init_cond;
% Weapon-ammunition system parameters
par=projectile_parameters;
% Trajectory calculation
[tr]=real_trajectory_num(ci,par);
[imp]=impact_point(tr)
[sag]=apex(tr);
figure
plot(tr.x,tr.y)
hold on
grid on
plot(ci.X0,ci.Y0,'or')
plot(imp.x,imp.y,'*r')
plot(sag.x,sag.y,'sr')
xlabel('X[m]')
ylabel('Y[m]')
title('Trajectory plot')
```

3. Concrete Example: Real vs. Ideal Trajectories

3.1. Parameter Table

Parameter	Symbol	Value
Initial speed	V_0	800 m/s
Launch angle	θ_0	45°
Mass	m	0.425 kg
Cross-section area	S	0.0045 m²
Sea-level density	ρ_0	1.225 kg/m³
Scale height	H	8000 m

3.2. Results

- Ideal (no drag): Range ≈ 65 km; Apex ≈ 16.3 km
- Real (with drag): Range ≈ 60 km; Apex ≈ 14.6 km

3.3. Plot Examples

- Trajectories overlaid
- Annotated apex and impact points

4. Activities and Exercises

Vary Launch Angle

In `init_cond.m`, set `theta0 = [10, 20, 45]`.

Run simulations and tabulate angle vs. range vs. max height.

Mass Sensitivity

- Change `m` from 0.425 kg to 1 kg.
- Observe how trajectory changes and discuss drag's dependence on mass.

5. Ideal vs. Real Trajectory Comparison and Plotting

To highlight drag effects, compare real and ideal trajectories in a single figure.

- **Real vs. Ideal:** The real trajectory falls shorter and lower than the ideal due to drag.
- **Impact Points:** Markers show ground impact for both models.
- **Apex:** Triangles denote maximum heights, illustrating performance loss.

6. Numerical Results Examples

Running the simulation for `theta0 = 45°`, `m = 0.425 kg`, `V0 = 800 m/s` yields:

Model	Angle (°)	Range (m)	Max Height (m)
Real	45	60032	14620
Ideal	45	65026	16327

For angles `[10°, 20°, 45°]`, the real trajectory results may be:

Angle (°)	Range (m)	Max Height (m)
10	14450	1970
20	27400	5330
45	60032	14620

Task I:

```
%initial_cond.m
% Returns a struct ci with initial conditions:
%   ci.X0     – initial horizontal position [m]
%   ci.Y0     – initial vertical position [m]
%   ci.V0     – initial speed [m/s]
%   ci.theta0 – launch angle [rad]
```

```
% No inputs. Hard-coded values.
```

```
% projectile_parameters.m
% Returns a struct par with projectile parameters:
%   par.d   – diameter [m]
%   par.m   – mass [kg]
%   par.ind – Siacci form factor (dimensionless)
%   par.g   – gravitational accel. [m/s^2]
% No inputs. Hard-coded values.
```

```
% real_trajectory_model.m
% function dW = real_trajectory_model (t, W, par)
%   Calculates time-derivatives [dV; dθ; dx; dy] given:
%     W = [V; θ; x; y] at time t
%     par = projectile params
%   Includes aerodynamic drag via cxsiacci and density ro(y).
```

```
% real_trajectory_num.m
% function tr = real_trajectory_num (ci, par)
%   Integrates model_traiect_real from t=0 until y=0 (impact).
%   Inputs:
%     ci – initial conditions struct
%     par – parameters struct
%   Outputs tr struct with fields:
%     tr.t – time vector
%     tr.V, tr.theta – V(t), θ(t)
%     tr.x, tr.y     – trajectory coordinates
%     tr.n           – number of steps
```

```
% impact_point.m
% function imp = impact_point (tr)
%   Finds final point of impact:
%     imp.x = tr.x(end)
%     imp.y = 0
%   No inputs except tr; no loops.
```

```
% apex.m
% function sag = apex. (tr)
%   Locates apex of trajectory:
%     [~, kmax] = max(tr.y);
%     sag.x = tr.x(kmax);
%     sag.y = tr.y(kmax);
```

1.2 Data-Flow Diagram

- **`init_cond`** and **`projectile_parameters`** produce **ci** and **par**.
- **`real_trajectory_num (ci,par)`** calls internally **`real_trajectory_model`** to build **tr**.

- `impact_point(tr)` extracts impact point **imp**.
- `apex(tr)` extracts apex **sag**.
- `trajectory_animation` plots **tr**, **imp**, and **sag** in an animation loop.

Task II:

```
% task_param_variation.m
% Vary θ₀, mass m, and frontal area S (via diameter d) one at a time, compute
range & max height, then plot results. Load base initial conditions & params
ci_base  = init_cond;
par_base = projectile_parameters;
%% 1) Vary launch angle θ₀
angles_deg = linspace(5, 85,  20);  % 20 values from 5° to 85°
angles = deg2rad(angles_deg);
range_theta  = zeros(size(angles));
height_theta = zeros(size(angles));
for i = 1:numel(angles)
    ci = ci_base;
    ci.theta0 = angles(i);
    tr = real_trajectory_num(ci, par_base);
    imp = impact_point(tr);
    sag = apex(tr);
    range_theta(i)  = imp.x;
    height_theta(i) = sag.y;
end
%% 2) Vary mass m
m_vals = linspace(par_base.m*0.5, par_base.m*2, 20);  % from 50% to 200% of base
range_m  = zeros(size(m_vals));
height_m = zeros(size(m_vals));
for i = 1:numel(m_vals)
    par = par_base;
    par.m = m_vals(i);
    tr = real_trajectory_num(ci_base, par);
    imp = impact_point(tr);
    sag = apex(tr);
    range_m(i)  = imp.x;
    height_m(i) = sag.y;
end
%% 3) Vary frontal area S via diameter d and compute base S
S_base = pi * par_base.d^2 / 4;
% choose S from 50% to 200% of base, then back-compute d for each
S_vals = linspace(S_base*0.5, S_base*2, 20);
d_vals = sqrt(4*S_vals/pi);
range_S  = zeros(size(S_vals));
height_S = zeros(size(S_vals));
for i = 1:numel(d_vals)
    par = par_base;
    par.d = d_vals(i);
    tr = real_trajectory_num(ci_base, par);
    imp = impact_point(tr);
    sag = apex(tr);
```

```
    range_S(i)  = imp.x;
    height_S(i) = sag.y;
end
figure('Position',[100 100 900 700]);
%  ┌──────────────┐
%  │ θ₀ variation │
%  └──────────────┘
subplot(3,2,1);
plot(angles_deg, range_theta, 'b-o','LineWidth',1.5);
xlabel('θ₀ [°]'); ylabel('Range [m]');
title('Range vs. Launch Angle');
subplot(3,2,2);
plot(angles_deg, height_theta, 'r-s','LineWidth',1.5);
xlabel('θ₀ [°]'); ylabel('Max Height [m]');
title('Height vs. Launch Angle');
%  ┌──────────────┐
%  │ m variation  │
%  └──────────────┘
subplot(3,2,3);
plot(m_vals, range_m, 'b-o','LineWidth',1.5);
xlabel('Mass m [kg]'); ylabel('Range [m]');
title('Range vs. Mass');
subplot(3,2,4);
plot(m_vals, height_m, 'r-s','LineWidth',1.5);
xlabel('Mass m [kg]'); ylabel('Max Height [m]');
title('Height vs. Mass');
%  ┌──────────────┐
%  │ S variation  │
%  └──────────────┘
subplot(3,2,5);
plot(S_vals, range_S, 'b-o','LineWidth',1.5);
xlabel('Frontal Area S [m^2]'); ylabel('Range [m]');
title('Range vs. Frontal Area');
subplot(3,2,6);
plot(S_vals, height_S, 'r-s','LineWidth',1.5);
xlabel('Frontal Area S [m^2]'); ylabel('Max Height [m]');
title('Height vs. Frontal Area');
sgtitle('Task 1 – Sensitivity to θ₀, m, and S','FontSize',14);
```