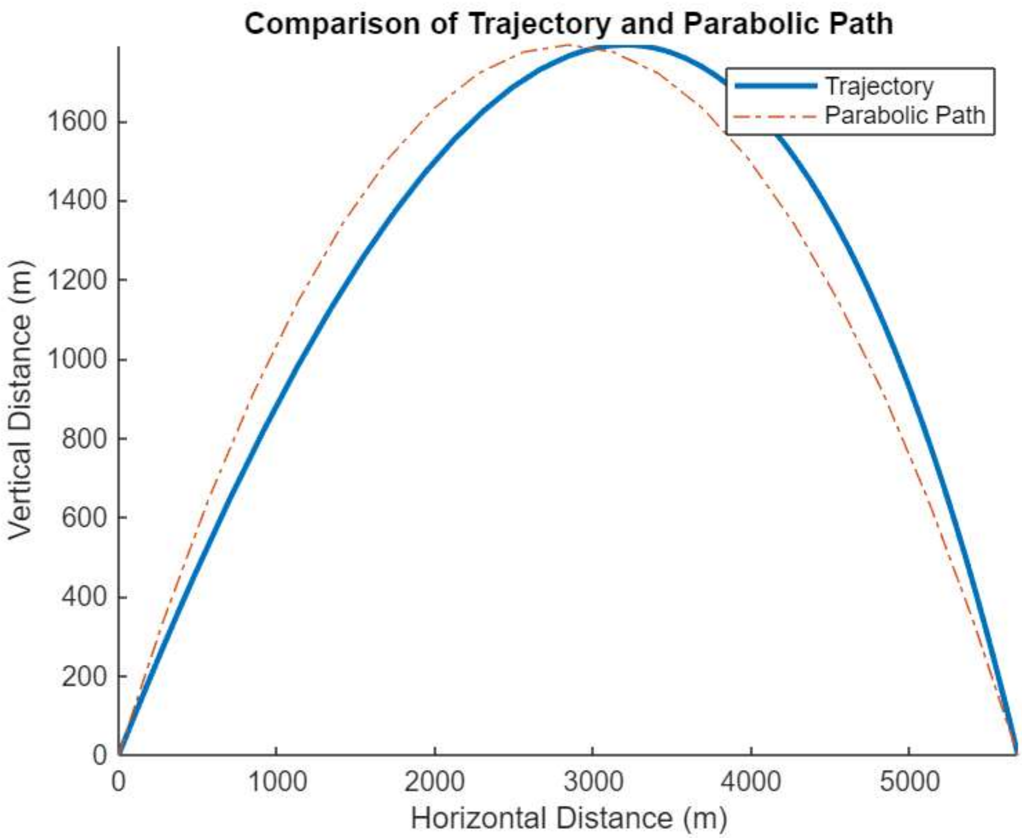
```
params.Theta = 45;       % Launch angle in degrees
params.Mu = 0.02;        % Air resistance coefficient
params.Model = "Stokes"; % Air resistance model: "None", "Stokes", or "Newton"

[maxHeight, maxRange, tMaxHeight, tMaxRange] = AirResistanceFunction(params);
```




```
function [maxHeight,maxRange,tMaxHeight,tMaxRange] = AirResistanceFunction(params)
    % Extract parameters
    % Here, I'm setting up the input parameters from the `params` structure.
    % `theta` is the launch angle, `mu` is the coefficient of air resistance,
    % and I'm using `g` for gravity and `vInitial` for the initial velocity.
    theta = params.Theta; % Launch angle in degrees
    mu = params.Mu;       % Air resistance coefficient
    g = 9.81;             % Gravitational acceleration in m/s²
    vInitial = 300;       % Initial velocity in m/s

    % Define time bounds
    % I'm calculating the total flight time for the projectile without air resistance
    % as an estimate for the simulation duration. I added 1 second to ensure full capture.
    tInitial = 0;
    tFinal = 2 * vInitial * sind(theta) / g + 1;

    % Define initial conditions
    % Initial position is (0,0) with horizontal and vertical velocity components split
    % using cosd and sind for the angle in degrees. This ensures correct trigonometric calculations.
    yInitial = [0; 0; vInitial * cosd(theta); vInitial * sind(theta)];

    % Choose the air resistance model
    % Here, I'm using a `switch` to define the differential equations based on the air resistance model.
    % For no resistance, forces only include gravity. For "Stokes" and "Newton," I include velocity-dependent terms.
    switch params.Model
        case "None"
            % No air resistance
            dydt = @(t,y) [y(3); y(4); 0; -g];
        case "Stokes"
            % Stokes drag: air resistance proportional to velocity
            dydt = @(t,y) [y(3); y(4); -mu * y(3); -g - mu * y(4)];
        case "Newton"
            % Newton drag: air resistance proportional to velocity squared
            dydt = @(t,y) [y(3); y(4); -mu * y(3) * sqrt(y(3)^2 + y(4)^2);
```

```
            dydt = @(t,y) [y(3); y(4); -mu * y(3) * sqrt(y(3)^2 + y(4)^2); ...
                           -g - mu * y(4) * sqrt(y(3)^2 + y(4)^2)];
        otherwise
            % Error handling for invalid models
            error("Invalid air resistance model");
    end

    % Solve for maximum height
    % I'm using `ode45` to integrate the equations until the projectile reaches the peak.
    options = odeset('Events', @endOfAscent); % Stop integration at the peak
    [~, yout, te, ye, ~] = ode45(dydt, [tInitial tFinal], yInitial, options);
    tMaxHeight = te;      % Time to reach max height
    maxHeight = ye(2);    % Maximum height

    % Solve for maximum range
    % After reaching the peak, I update the initial conditions and continue integration until the projectile hits the ground.
    tInitial = te;
    yInitial = ye';
    options = odeset('Events', @endOfDescent); % Stop integration when it hits the ground
    [~, y, te, ye, ~] = ode45(dydt, [tInitial tFinal], yInitial, options);
    yout = [yout; y(2:end,:)]; % Combine the results
    tMaxRange = te;           % Time to reach max range
    maxRange = ye(1);         % Maximum range

    % Plotting the results
    % I want to visualize the trajectory and compare it to an ideal parabolic path.
    figure(Name="Projectile Trajectory");
    hold on;
    % Plot the actual trajectory
    plot(yout(:,1), yout(:,2), 'LineWidth', 2);
    % Plot the parabolic path for comparison
    X = maxRange * (0:0.05:1);
    Y = 4 * maxHeight * X .* (maxRange - X) / maxRange^2;
    plot(X, Y, '-.');
    % Add titles, labels, and a legend for clarity
    title("Comparison of Trajectory and Parabolic Path");
    xlabel("Horizontal Distance (m)");
    ylabel("Vertical Distance (m)");
    legend("Trajectory", "Parabolic Path");
    axis tight;
    hold off;
end

% Helper function: Detect when the projectile reaches its peak
function [value, isterminal, direction] = endOfAscent(~, y)
    % I stop integration when the vertical velocity (y(4)) is zero.
    value = y(4);         % Vertical velocity
    isterminal = 1;       % Stop the integration
    direction = 0;        % Detect zero crossing in any direction
end

% Helper function: Detect when the projectile hits the ground
function [value, isterminal, direction] = endOfDescent(~, y)
    % I stop integration when the height (y(2)) is zero.
    value = y(2);         % Height
    isterminal = 1;       % Stop the integration
    direction = 0;        % Detect zero crossing in any direction
end
```