Simulation of Projectile Trajectories using Forward Euler Method

Tomasz Karpiński

October 14, 2025

Abstract

This report summarises numerical simulations of projectile motion performed with the Forward Euler method. The model includes aerodynamic drag with and without altitude correction. Results include convergence tests (no drag), the influence of drag on trajectories and range, scans of firing angle vs range for several drag coefficients, and a comparison of altitude correction effects for artillery-like parameters.

1 Introduction

We study the two-dimensional motion of an inert projectile launched with initial speed v_0 and angle θ_0, subject to gravity and aerodynamic drag. The total force acting on the projectile is

$$\mathbf{F} = \mathbf{F}_{\text{grav}} + \mathbf{F}_{\text{drag}},$$

where

$$\mathbf{F}_{\text{grav}} = -mg\,\hat{\mathbf{y}}, \qquad \mathbf{F}_{\text{drag}} = -D\,v\,\mathbf{v}\,\frac{\rho(y)}{\rho_0}.$$

Here m is the projectile mass, g is gravitational acceleration, D is the drag factor, $\mathbf{v} = (v_x, v_y)$ is velocity, $v = \|\mathbf{v}\|$, and $\rho(y)$ is the altitude-dependent air density used when altitude correction is enabled. Using Newton's second law the equations of motion are:

$$\begin{aligned} \ddot{x}(t) &= -\frac{D}{m}\,v\,v_x\,\frac{\rho(y)}{\rho_0}, \\ \ddot{y}(t) &= -g - \frac{D}{m}\,v\,v_y\,\frac{\rho(y)}{\rho_0}. \end{aligned} \tag{1}$$

Without altitude correction one sets $\rho(y)/\rho_0 = 1$. With adiabatic altitude correction:

$$\frac{\rho(y)}{\rho_0} = \left(1 - \frac{a\,y}{T_0}\right)^{\alpha},$$

where typical choices are $a = 6.5 \times 10^{-3}$ K/m, $T_0 = 293$ K and $\alpha = 2.5$.

2 Numerical method

2.1 General Forward Euler

For a general first-order ODE

$$\frac{du}{dt} = f(t, u), \qquad u(t_0) = u_0,$$

the Forward Euler discretisation with time-step Δt reads

$$u_{n+1} = u_n + \Delta t\; f(t_n, u_n).$$

This scheme is explicit and first-order accurate in time (global error $O(\Delta t)$).

2.2 Application to projectile equations

We transform the second-order system (1) into first-order form by introducing $v_x = \dot{x}$ and $v_y = \dot{y}$. The system to integrate is

$$\begin{aligned}
\dot{x} &= v_x, \\
\dot{v}_x &= -\frac{D}{m}\, v\, v_x\, \frac{\rho(y)}{\rho_0}, \\
\dot{y} &= v_y, \\
\dot{v}_y &= -g - \frac{D}{m}\, v\, v_y\, \frac{\rho(y)}{\rho_0}.
\end{aligned}$$

Applying Forward Euler to these equations yields the update formulas used in the implementation:

$$\begin{aligned}
x_{n+1} &= x_n + \Delta t\, v_{x,n}, \\
v_{x,n+1} &= v_{x,n} + \Delta t \left(-\frac{D}{m}\, v_n\, v_{x,n}\, \frac{\rho(y_n)}{\rho_0} \right), \\
y_{n+1} &= y_n + \Delta t\, v_{y,n}, \\
v_{y,n+1} &= v_{y,n} + \Delta t \left(-g - \frac{D}{m}\, v_n\, v_{y,n}\, \frac{\rho(y_n)}{\rho_0} \right).
\end{aligned}$$

2.3 Algorithm outline and implementation

The implementation follows the Forward Euler loop until the projectile crosses the ground $y = 0$. When a crossing is detected (i.e. $y_{n+1} < 0$), the code computes the fractional part r of the last time-step at which the ground was reached by linear interpolation and corrects x, y, t so that the final altitude is exactly zero. The simulation data (time, x, y) is written to a CSV file.

Below is the exact C++ function used to generate trajectories (this is the core of the numerical algorithm):

```cpp
// runSimulation: compute trajectory and save t,x,y to CSV; returns numerical x_max
double runSimulation(double v0, double theta0, double delta_t,
                     double D, double a, double m, double alpha,
                     const std::string& filename) {

    double t = 0.0;
    const double g = 9.81;
    const double T0 = 293;

    // at the beginning
    double x = 0.0;
    double y = 0.0;

    // velocities
    double vx = v0 * std::cos(theta0);
    double vy = v0 * std::sin(theta0);

    bool RUN = true;

    std::ofstream results(filename);
    if (!results.is_open()) {
        std::cerr << "Error with opening the file: " << filename << std::endl;
        return -1.0;
    }

    double x_final = 0.0;

    do {
        double v = sqrt(vx * vx + vy * vy);

        // save for collision correction
        double x_old = x;
        double y_old = y;
        double t_old = t;

        // calculating new position
        // Fx = Eq.15, Fy = Eq.15
        double Fx = -D * v * vx * rho(y, a, T0, alpha);
        double Fy = -D * v * vy * rho(y, a, T0, alpha);

        // Euler formula
        x = x + delta_t * vx;
        vx = vx + delta_t * (Fx / m);
        y = y + delta_t * vy;
        vy = vy + delta_t * ((Fy / m) - g);

        // time step
        t = t + delta_t;

        // detect the collision + correct position
        if (y < 0.0) {
            // the fraction of time step after which the collision will be registered
            double r = y_old / (y_old - y);

            // position correction
            x = x_old + (x - x_old) * r;
            y = y_old + (y - y_old) * r;

            // time correction
            t = t_old + (t - t_old) * r; // Use t_old and new t to correct time

            RUN = false; // stop
        }

        results << t << ", " << x << ", " << y << std::endl;
        x_final = x;

    } while (RUN);

    results.close();
    return x_final; // numerical x_max
}
```

3 Results

3.1 Convergence of trajectories and global error (no drag)

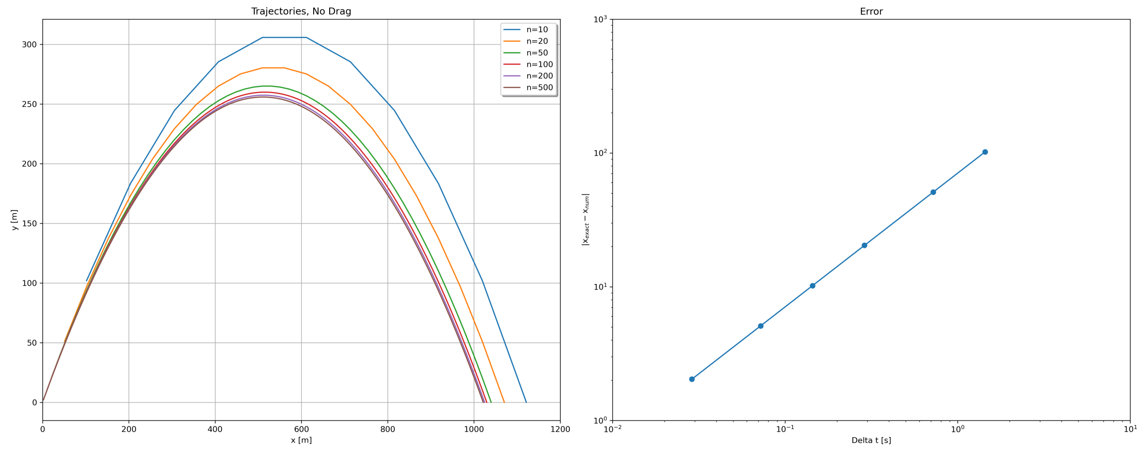

Figure 1: Left: numerical trajectories for several time-step resolutions (no drag). Right: global error $E_{\text{glob}} = |x_{\text{exact}} - x_{\text{num}}|$ vs Δt in log–log scale.

Table 1: Global error vs time step (no drag). Values taken from program output.

n	Δt [s]	x_{exact} [m]	x_{num} [m]	E_{glob} [m]
10	1.4416	1019.37	1121.30	101.937
20	0.720802	1019.37	1070.34	50.9684
50	0.288321	1019.37	1039.76	20.3874
100	0.14416	1019.37	1029.56	10.1937
200	0.0720802	1019.37	1024.46	5.09684
500	0.0288321	1019.37	1021.41	2.03874

Observations: The left panel shows that trajectories approach the analytic parabola as Δt decreases. The log–log error plot has slope close to 1, consistent with the first-order accuracy of Forward Euler. Practically, for a target accuracy in range one must choose sufficiently small Δt (e.g. $n \geq 200$ in our tests gives errors on the order of a few meters).

3.2 Trajectories for different drag coefficients (no altitude correction)

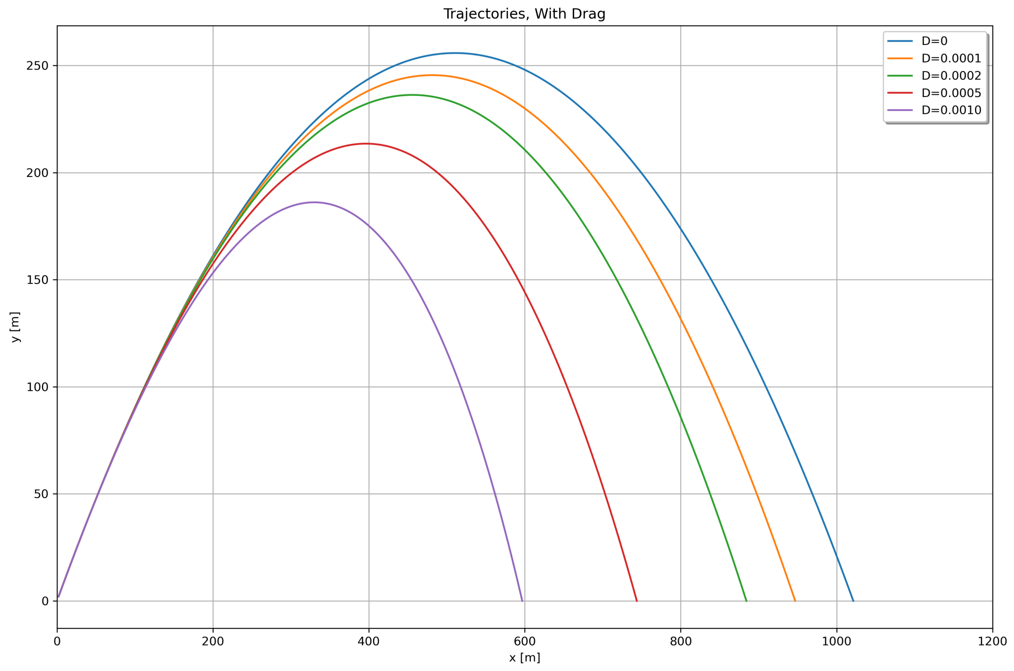

Figure 2: Trajectories for several drag coefficients D with altitude correction disabled ($a = 0$).

Table 2: Range vs drag coefficient for $v_0 = 100$ m/s, $\theta_0 = 45°$, $m = 1$ kg.

D	range $x_{\max}$ [m]
0	1021.41
1×10^{-4}	946.88
2×10^{-4}	884.258
5×10^{-4}	743.719
1×10^{-3}	596.775

Observations: Increasing D reduces both maximum height and horizontal range. The effect is nonlinear — a small increase in drag at these speeds yields a noticeable reduction of range. For applied tasks this confirms the necessity to include drag for realistic predictions.

3.3 Range as a function of firing angle for several drag coefficients

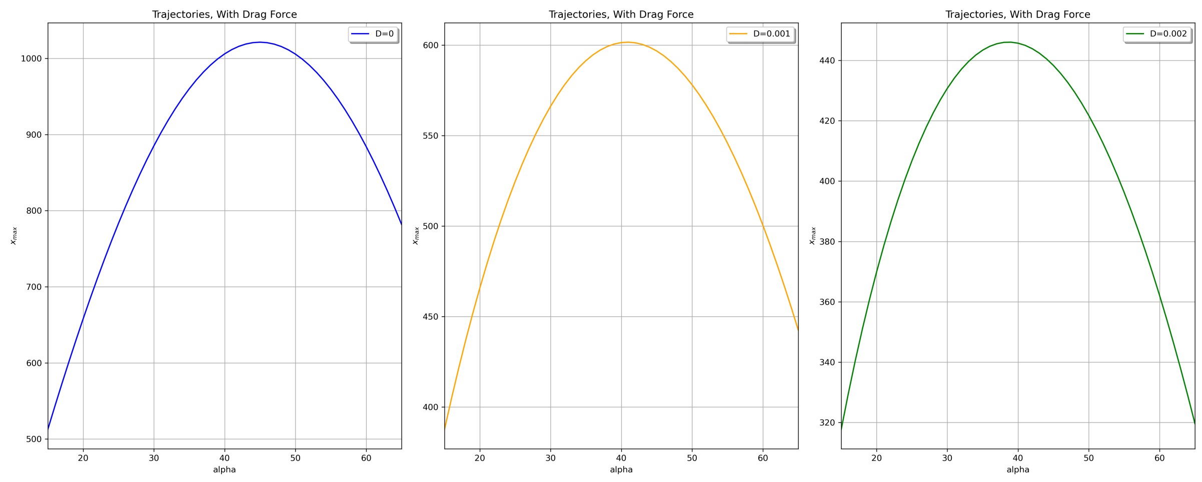

Figure 3: Range vs firing angle panels for $D = 0$ (left), $D = 10^{-3}$ (centre), and $D = 2 \times 10^{-3}$ (right).

Table 3: Selected maxima from angle scans: optimal angle and corresponding range.

D	θ_{opt} [deg]	$x_{\max}$ [m]
0	45	1021.41
0.001	41	601.65
0.002	39	446.05

Observations: For no drag, classical ballistic theory holds and the optimal angle is 45°. With increasing drag the optimal angle decreases (becomes flatter), because steeper trajectories suffer greater losses due to higher air resistance integrated over time.

3.4 Drag with altitude correction (high-speed, artillery-like conditions)

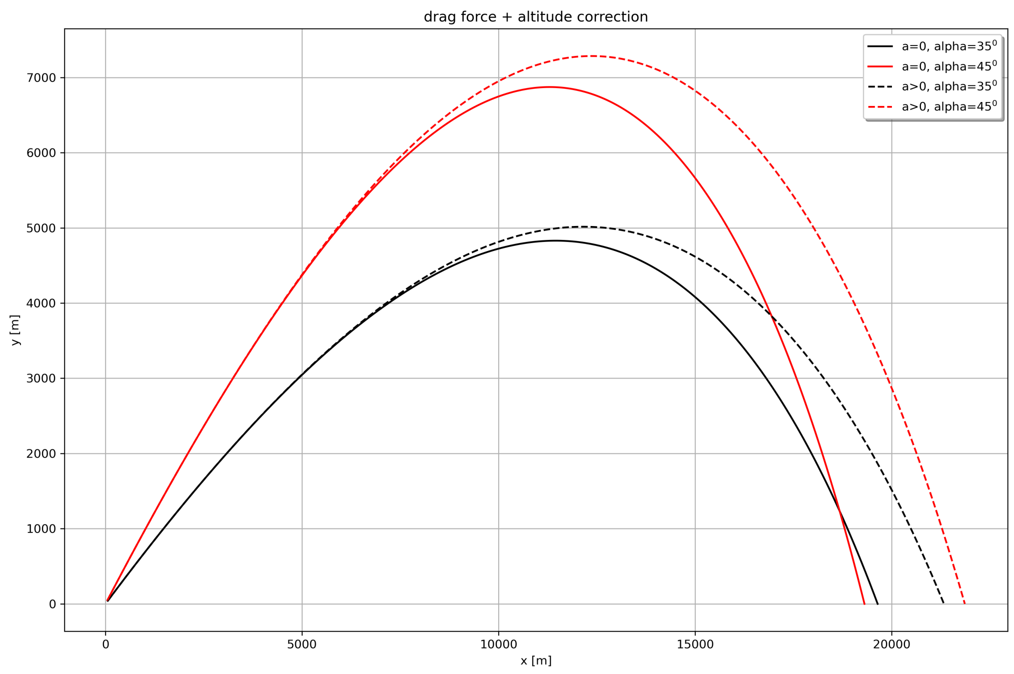

Figure 4: Trajectories for $\theta_0 = 35°, 45°$ with and without altitude correction.

Observations: Enabling altitude correction reduces the effective drag at high altitude and thus slightly increases apogee and horizontal range for the same initial conditions. This effect becomes visible for the large initial speed and mass used here: the relative difference is moderate but clearly measurable in the computed trajectories.

4 Conclusions

This project implemented a Forward Euler integrator for projectile motion including drag and optional altitude-dependent density. Key findings:

- The Forward Euler method shows first-order convergence in time: global error scales approximately linearly with Δt. For acceptable accuracy in range computations one should use relatively small timesteps (e.g. $\Delta t = t_{\text{flight}}/500$ for the presented tests).
- Aerodynamic drag strongly reduces range and maximum altitude; the dependence on drag is nonlinear and significant even for small D.
- The optimal launch angle for maximum range shifts from 45° (no drag) towards smaller angles as drag increases.
- Altitude correction (adiabatic density decrease) reduces drag at higher altitudes and yields modest increases in range and apogee for high-speed/heavy projectiles.