Analytical Approximations of Projectile Motion for Linear and Quadratic Air Drag

Richard Saucier

August 2012

Abstract

The purpose of this document is to provide closed-form solutions to projectile motion while accounting for, or approximating, the effect of air resistance. The plan is to first quickly review the solution when air resistance is neglected, and then to approximate the effect of air drag. Making the assumption that the drag force varies linearly with air speed allows us to solve the equations of motion exactly. The more realistic assumption that air drag varies as the square of the speed can also be solved analytically, provided that we restrict it to shallow launch angles. This analytical approximate solution is compared to the exact numerical solution using a fourth-order Runge-Kutta method.

Contents

List of Figures

List of Listings

1 Introduction

It is well known that projectile motion under the influence of gravity and neglecting air resistance is a problem that can be solved analytically. The closed-form solution is easily manipulated to provide range, maximum range, projectile path, height, time of flight, impact velocity, and angle of impact, among others. In short, we know everything about the motion.

In the more realistic case when air resistance is taken into account, the problem is much harder to deal with. As a result, the usual strategy is to solve the equations numerically—for example using the Runge-Kutta method. This is straightforward enough, but numerical solutions do not provide the same level of insight that we get from analytical formulas. The relatively recent introduction of the Lambert W function [4] and its implementation in computer code [5] provides a tool that can be applied to this situation. Although it will not allow us to solve the problem in its full generality, this function will facilitate analytical solutions under reasonable approximations.

The purpose of this report is to provide closed-form solutions to projectile motion while accounting for, or approximating, the effect of air resistance. The plan is to first quickly review the solution when air resistance is neglected, and then to approximate the effect of air drag. Making the assumption that the drag force varies linearly with air speed allows us to solve the equations of motion exactly. The more realistic assumption that air drag varies as the square of the speed can also be solved analytically, provided we restrict it to shallow launch angles. This analytical solution is compared to the numerical solution using a fourth-order Runge-Kutta method.

2 Projectile Motion in the Absence of Air Drag

We review the equations of motion for a projectile launched from the ground and falling under the influence of gravity but neglecting air resistance. There is nothing new here, but it does serve to introduce our notation and establish some results that will be useful later.

Thus, let $\hat{\boldsymbol{\imath}}$ and $\hat{\boldsymbol{\jmath}}$ be unit vectors along the x (horizontal) and y (vertical) axes, respectively. Then the force acting on the projectile of mass m can be written as $\boldsymbol{F} = -mg\hat{\boldsymbol{\jmath}}$, where g is the acceleration due to gravity. Also, let u be the velocity along the x-axis and w be the velocity along the y-axis, so that $\boldsymbol{v} = u\hat{\boldsymbol{\imath}} + w\hat{\boldsymbol{\jmath}}$ is the total vector velocity of the projectile. Then from Newton's second law, the vector equation of motion,

$$m\frac{d\boldsymbol{v}}{dt} = -mg\hat{\boldsymbol{\jmath}}, \tag{1}$$

can be resolved in terms of its two components,

$$\dot{u} = 0 \quad \text{and} \quad \dot{w} = -g, \tag{2}$$

where dot notation is used to signify derivatives with respect to time. Integrating these equations gives the velocities as a function of time,

$$u = u_0 \quad \text{and} \quad w = w_0 - gt, \tag{3}$$

where u_0 and w_0 are the initial velocities along the x-axis and y-axis respectively. In terms of the launch angle, θ, as measured from the horizontal, $u_0 = v_0\cos\theta$ and $w_0 = v_0\sin\theta$, where v_0 is the launch speed. Integrating again gives the location as a function of time,

$$x = (v_0\cos\theta)\,t \quad \text{and} \quad y = (v_0\sin\theta)\,t - \tfrac{1}{2}gt^2. \tag{4}$$

Eliminating t between these two equations then gives the equation for the trajectory,

$$y = \left(\frac{\sin\theta}{\cos\theta}\right)x - \left(\frac{g}{2v_0^2\cos^2\theta}\right)x^2. \tag{5}$$

By completing the square in x, this can also be written as

$$y = \frac{v_0^2 \sin^2\theta}{2g} - \frac{g}{2v_0^2\cos^2\theta}\left(x - \frac{v_0^2 \sin 2\theta}{2g}\right)^2, \tag{6}$$

which is recognized as the equation for a parabola with its apex (maximum) at $x = (v_0^2 \sin 2\theta)/2g$ and $y = (v_0^2 \sin^2\theta)/2g$. The trajectory is symmetric about this point, which means the range is $(v_0^2 \sin 2\theta)/g$ and the height is $(v_0^2 \sin^2\theta)/2g$. The range, R, can also be obtained directly from eq. (5) as the non-trivial value of x that gives $y = 0$:

$$R = \frac{v_0^2}{g}\sin 2\theta. \tag{7}$$

By setting $dR/d\theta = 0$ and solving for θ, or simply by examining $\sin 2\theta$, the maximum range is found to be when the firing angle is 45°, and its value is

$$R_{\max} = \frac{v_0^2}{g}. \tag{8}$$

For a given initial velocity and firing angle, the projectile reaches its maximum height, H, when $w = 0$ and this occurs when $t = v_0 \sin\theta/g$ from eq. (3). Substituting this into eq. (4) gives

$$H = \frac{v_0^2 \sin^2\theta}{2g}, \tag{9}$$

which agrees with the maximum of the parabolic path found previously. The total time of flight, T, is twice the time to reach its maximum height, so that

$$T = \frac{2v_0 \sin\theta}{g}. \tag{10}$$

The average speed over the entire flight is the range divided by the total time, $\bar{v} = R/T$, which gives

$$\bar{v} = v_0 \cos\theta. \tag{11}$$

Substituting the total flight time into eq. (3), the impact velocity is found to be

$$u = v_0\cos\theta \quad \text{and} \quad w = -v_0 \sin\theta, \tag{12}$$

and it is clear that

- the impact angle is the negative of the launch angle, and
- the impact speed is the same as the launch speed.

It is also clear that the motion only depends upon the launch speed, v_0, and launch angle, θ, and is independent of the mass. Typical projectile paths are shown in Figure 1. These results will serve as points of reference as we consider projectile motion under more realistic conditions of air resistance.

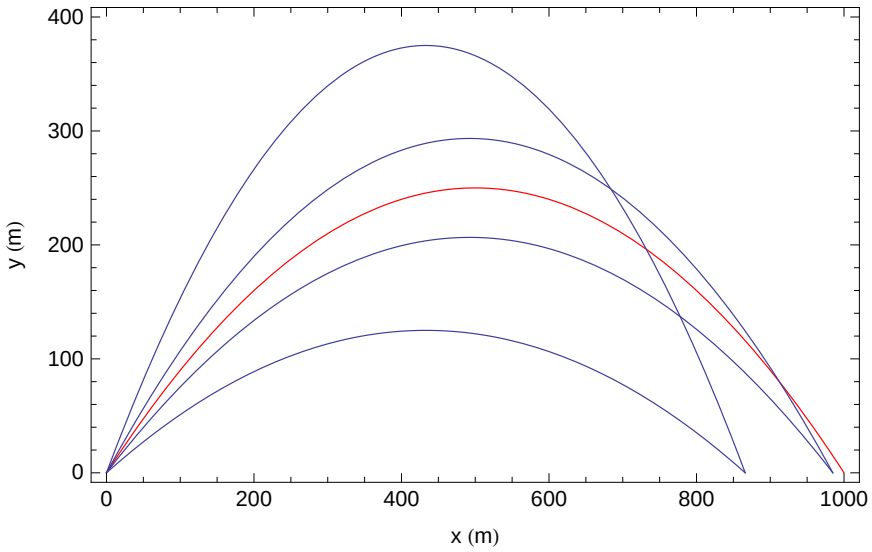

Figure 1. Trajectories neglecting air drag for launch angles of 30°, 40°, 50°, 60° (blue) and $\theta_{\max} = 45^\circ$ (red)

3 Projectile Motion with Linear Air Drag ($C_d \sim 1/\mathrm{Ma}$)

Now we begin to account for air resistance and determine its effect upon projectile motion. The drag force acting on a projectile due to air resistance is velocity dependent and is of the form

$$\boldsymbol{F}_d = -\tfrac{1}{2} C_d A \rho v^2 \hat{\boldsymbol{v}}, \tag{13}$$

where C_d is the (dimensionless) drag coefficient, A is the projectile presented area, ρ is the air density, v is the speed with respect to air, and $\hat{\boldsymbol{v}}$ is a unit vector in the direction of the velocity. The negatve sign indicates the force is resistive and opposes the motion. If the drag coefficient, C_d, is constant, then the drag force goes as the square of the speed. However, the drag coefficient is itself a complex function of velocity, [8] and if there is a portion of the C_d *vs* v drag curve that behaves as $1/v$, then we have a net linear air-drag force. [9] This is most easily expressed by saying that the drag coefficient goes as the inverse of the dimensionless Mach number, $\mathrm{Ma} \equiv v/a$, where a is the speed of sound in air. In any case, it is useful to first solve the equations of motion for linear air resistance as a prelude to the more complex quadratic air drag.

So let us consider the case where $C_d \sim 1/\mathrm{Ma}$ and let $C_d \implies C_d/\mathrm{Ma}$. The vector equation of motion then has the form

$$\frac{d\boldsymbol{v}}{dt} = -b\boldsymbol{v} - g\hat{\boldsymbol{\jmath}}, \tag{14}$$

where the coefficient

$$b \equiv \frac{\rho a C_d A}{2m} \tag{15}$$

has dimensions of s^{-1}. In terms of the two components, the equations of motion are

$$\dot{u} = -bu \quad \text{and} \quad \dot{w} = -bw - g. \tag{16}$$

The solutions are

$$u = u_0 e^{-bt} \tag{17}$$

for the x component of the velocity and

$$x = \frac{u_0}{b}\left(1 - e^{-bt}\right) \tag{18}$$

for the downrange location (using the origin as the starting location).[1] The y component of the velocity is

$$w = \left(w_0 + \frac{g}{b}\right) e^{-bt} - \frac{g}{b} \tag{19}$$

and

$$y = \left(w_0 + \frac{g}{b}\right) \frac{1}{b} \left(1 - e^{-bt}\right) - \frac{gt}{b} \tag{20}$$

for the vertical position of the projectile (again using the origin as the starting position). Eliminating t between eqs. (17) and (19), gives the equation for the trajectory,

$$y = \frac{g}{b^2} \left[\left(1 + \frac{bw_0}{g}\right) \frac{bx}{u_0} + \ln\left(1 - \frac{bx}{u_0}\right)\right]. \tag{21}$$

It is instructive to compare this trajectory equation to its counterpart which neglected air drag, eq. (6). Absence of air drag results in y as a simple quadratic function of x. With linear air drag, x appears in a non-algebraic way, involving both x and its logarithm.

The range, R, is the distance downrange when $y = 0$, so it is the non-trivial solution to the equation

$$\left(1 + \frac{bw_0}{g}\right) \frac{bR}{u_0} + \ln\left(1 - \frac{bR}{u_0}\right) = 0. \tag{22}$$

This is a transcendental equation for R that, until recently, could not be solved algebraically but had to be solved numerically. With the definition of the Lambert W function[2] as the solution of the equation

$$W(z)e^{W(z)} = z, \tag{23}$$

we now have a way of solving these types of equations. [6, 10] The strategy is to rearrange it into the form of eq. (23) with a known quantity on the right-hand side to play the role of z, then we simply identify $W(z)$ as the Lambert W function.

We show the steps here. First, exponentiate both sides of eq. (21) to give

$$\exp\left[\left(1 + \frac{bw_0}{g}\right) \frac{bR}{u_0}\right] \left(1 - \frac{bR}{u_0}\right) = 1, \tag{24}$$

and

$$\left(1 + \frac{bw_0}{g}\right) \left(\frac{bR}{u_0} - 1\right) \exp\left[\left(1 + \frac{bw_0}{g}\right) \frac{bR}{u_0}\right] = -\left(1 + \frac{bw_0}{g}\right), \tag{25}$$

and finally

$$\left(1 + \frac{bw_0}{g}\right) \left(\frac{bR}{u_0} - 1\right) \exp\left[\left(1 + \frac{bw_0}{g}\right) \left(\frac{bR}{u_0} - 1\right)\right] = -\left(1 + \frac{bw_0}{g}\right) \exp\left[-\left(1 + \frac{bw_0}{g}\right)\right]. \tag{26}$$

Comparing this to the Lambert equation, eq. (23) we see that

$$W\left(-\left(1 + \frac{bw_0}{g}\right) \exp\left[-\left(1 + \frac{bw_0}{g}\right)\right]\right) = \left(1 + \frac{bw_0}{g}\right) \left(\frac{bR}{u_0} - 1\right). \tag{27}$$

[1] Notice that combining eqs. (16) (17) leads to the simple relationship $u = u_0 - bx$, which means that for shallow launch angles, the velocity is a linearly-decreasing function of range. This is a key observation of Celmins, who shows how this can be exploited to determine the drag coefficient. [9]

[2] "On the Lambert W function," R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey and D. E. Knuth *Advances in Computational Mathematics*, 1996, Volume 5, Number 1, pp. 329-359. Also available online at https://www.cs.uwaterloo.ca/research/tr/1993/03/W.pdf.

Solving this for R gives

$$R = \frac{u_0}{b}\left[1 + \frac{W(z)}{1 + bw_0/g}\right], \tag{28}$$

or

$$R = \frac{v_0 \cos\theta}{b}\left[1 + \frac{W(z)}{1 + \alpha \sin\theta}\right], \tag{29}$$

where $W(z)$ is the Lambert function[3],

$$z \equiv -(1 + \alpha \sin\theta)\exp[-(1 + \alpha \sin\theta)] \quad \text{and} \quad \alpha \equiv bv_0/g. \tag{30}$$

3.1 Maximum Range

Now, we know that the angle that maximizes the range when there is no air resistance is 45°. To find the corresponding angle in the case of linear air resistance, we set the derivative of R with respect to θ equal to zero and solve for θ:

$$\left(\frac{dR}{d\theta}\right)_{\theta_{\max}} = 0.$$

Implicit differentiation of the Lambert equation, $We^W = z$, gives

$$\frac{dW}{dz} = \frac{e^{-W}}{1 + W} = \frac{W}{(1 + W)z}.$$

Using this, along with the chain rule and eq. (29), we have

$$\frac{dW}{d\theta} = \frac{dW}{dz}\frac{dz}{d\theta} = \frac{W}{(1 + W)z}\frac{dz}{d\theta} = \frac{W}{1 + W}\frac{d\ln z}{d\theta} = \frac{W}{1 + W}\frac{-\alpha^2 \sin\theta\cos\theta}{1 + \alpha\sin\theta}. \tag{31}$$

So, from eq. (28), and using eq. (30),

$$\begin{aligned}
\frac{dR}{d\theta} &= -\frac{v_0 \sin\theta}{b}\left[1 + \frac{W}{1 + \alpha\sin\theta}\right] + \frac{v_0\cos\theta}{b}\frac{1}{1 + \alpha\sin\theta}\frac{W}{1 + W}\frac{-\alpha^2\sin\theta\cos\theta}{1 + \alpha\sin\theta} + \frac{v_0\cos\theta}{b}W\frac{-\alpha\cos\theta}{(1 + \alpha\sin\theta)^2} \\
&= -\frac{v_0\sin\theta}{b}\left[1 + \frac{W}{1 + \alpha\sin\theta}\right] - \frac{v_0\cos\theta}{b}\frac{W\alpha\cos\theta}{(1 + \alpha\sin\theta)^2}\left[\frac{\alpha\sin\theta}{1 + W} + 1\right] \\
&= -\frac{v_0}{b}\frac{1 + W + \alpha\sin\theta}{(1 + \alpha\sin\theta)^2}\left[\sin\theta(1 + \alpha\sin\theta) + \frac{W}{1 + W}\alpha(1 - \sin^2\theta)\right] \\
&= -\frac{v_0}{b}\frac{1 + W + \alpha\sin\theta}{(1 + \alpha\sin\theta)^2}\left[\frac{\alpha}{1 + W}\sin^2\theta + \sin\theta + \frac{\alpha W}{1 + W}\right] \\
&= -\frac{v_0}{b}\frac{1 + W + \alpha\sin\theta}{(1 + \alpha\sin\theta)^2}\frac{\alpha}{1 + W}\left[\sin^2\theta + \frac{1 + W}{\alpha}\sin\theta + W\right]
\end{aligned}$$

Requiring this to be zero gives the equation for $\theta_{\max}$, the firing angle to get the maximum range:

$$\sin^2\theta_{\max} + \frac{1 + W}{\alpha}\sin\theta_{\max} + W = 0. \tag{32}$$

Notice that, since $W(z)$ is a function of θ (see eq. 29), this is not a simple quadratic in $\sin\theta_{\max}$. We can solve it numerically—for example through bisection, since we know that the angle must lie somewhere in the

[3] Since R must be positive, we require that $W(z) > -(1 + \alpha\sin\theta)$ in eq. (29), which in the limit as $\alpha \to 0$ requires $W(z) > -1$. This implies that here we want $W_0(z)$, the principal branch of the Lambert W function (see Appendix A). Note that our focus has been on the range, but we haven't actually excluded the trivial solution of $x = 0$ when $y = 0$. Thus if we use x rather than R in eq. (29), this formula actually contains both solutions, $x = 0$ and $x = R$, due to the fact that $W(xe^x) = x$ is always the trivial solution for one of the branches. The author thanks Dr. Joseph Collins for pointing this out.

interval $(0, \pi/4)$—but a better option is to, once again, make use of the Lambert function to solve it explicitly. First, let $x \equiv \sin\theta_{\max}$, and solve eq. (31) for W,

$$W(z) = -\sin\theta_{\max}\frac{1+\alpha\sin\theta_{\max}}{\alpha+\sin\theta_{\max}} = -x\frac{1+\alpha x}{\alpha+x} \tag{33}$$

where, from eq. (29), $z = -(1+\alpha x)e^{-(1+\alpha x)}$. From the defining relation of the Lambert function, eq. (23), we know that

$$-x\frac{1+\alpha x}{\alpha+x}\exp\left[-x\frac{1+\alpha x}{\alpha+x}\right] = -(1+\alpha x)e^{-(1+\alpha x)}. \tag{34}$$

We want to solve this equation for x. Rearranging, we have

$$\frac{x}{x+\alpha}\exp\left[\frac{\alpha^2 x+\alpha}{x+\alpha}\right] = 1 \tag{35}$$

and

$$\frac{x}{x+\alpha}\exp\left[\frac{\alpha^2 x+x+\alpha-x}{x+\alpha}\right] = \frac{x}{x+\alpha}\exp\left[(\alpha^2-1)\frac{x}{x+\alpha}+1\right] = \frac{x}{x+\alpha}\exp\left[(\alpha^2-1)\frac{x}{x+\alpha}\right]e = 1,$$

so that

$$(\alpha^2-1)\frac{x}{x+\alpha}\exp\left[(\alpha^2-1)\frac{x}{x+\alpha}\right] = \frac{\alpha^2-1}{e},$$

and therefore,

$$W\left(\frac{\alpha^2-1}{e}\right) = (\alpha^2-1)\frac{x}{x+\alpha}.$$

Finally, solving this for $x = \sin\theta_{\max}$, we get

$$\theta_{\max} = \sin^{-1}\left[\frac{\alpha W\left(\frac{\alpha^2-1}{e}\right)}{\alpha^2-1-W\left(\frac{\alpha^2-1}{e}\right)}\right].$$

Since $W(0) = 0$ (which follows from eq. (23)), as it stands this expression has the indeterminate form $0/0$ when $\alpha = 1$. Returning to eq. (35) and setting $\alpha = 1$, we find that $x = 1/(e-1)$. Therefore, the complete solution for the maximum range launch angle is

$$\theta_{\max} = \begin{cases} \sin^{-1}\left(\dfrac{\alpha W\left(\frac{\alpha^2-1}{e}\right)}{\alpha^2-1-W\left(\frac{\alpha^2-1}{e}\right)}\right) & \text{if } \alpha \neq 1 \\ \sin^{-1}\left(\dfrac{1}{e-1}\right) & \text{if } \alpha = 1 \end{cases} \tag{36}$$

and this is shown plotted in Figure 2. The maximum range is obtained by substituting these values into eq. (29) while making use of eq. (33):

$$R_{\max} = \begin{cases} \dfrac{v_0^2}{g}\sqrt{\dfrac{\alpha^2-\left[1+W\left(\frac{\alpha^2-1}{e}\right)\right]^2}{\alpha^2(\alpha^2-1)}} & \text{if } \alpha \neq 1 \\ \dfrac{v_0^2}{g}\sqrt{\dfrac{e-2}{e}} & \text{if } \alpha = 1 \end{cases} \tag{37}$$

and where we used $v_0/b = (v_0^2/g)/\alpha$. Recall that the range at any angle θ in the case of no air resistance is $(v_0^2/g)\sin(2\theta)$. Let's now call this quantity R_0. Then, a natural comparison is $R_{\max}/R_0$, which is the

maximum range at the optimal launch angle *with air resistance*, as compared to the range at the same launch angle but *without air resistance*. Using eqs. 36 and 37, we get

$$\frac{R_{\max}}{R_0} = \begin{cases} \dfrac{\left(\alpha^2 - 1 - W\left(\frac{\alpha^2-1}{e}\right)\right)^2}{2\alpha^2(\alpha^2-1)W\left(\frac{\alpha^2-1}{e}\right)} & \text{if } \alpha \neq 1 \\ \dfrac{(e-1)^2}{2e} & \text{if } \alpha = 1 \end{cases} \tag{38}$$

and this is shown plotted in Figure 3. Both the ratio $R_{\max}/R_0$ and $\theta_{\max}$ are seen to be functions solely of the dimensionless parameter $\alpha \equiv bv_0/g$, which is itself the dimensionless ratio of the air drag deceleration to the acceleration due to gravity.

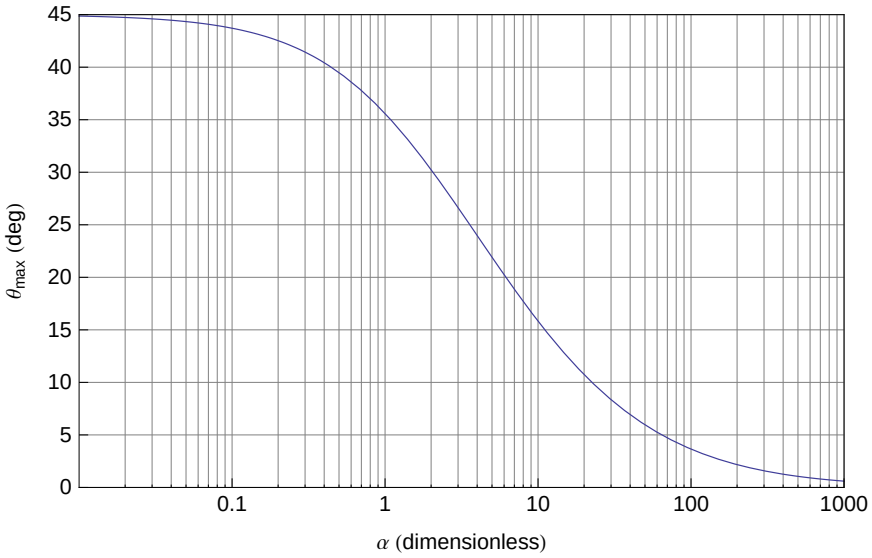

Figure 2. Firing angle to achieve maximum range as a function of the dimensionless parameter $\alpha = bv_0/g$

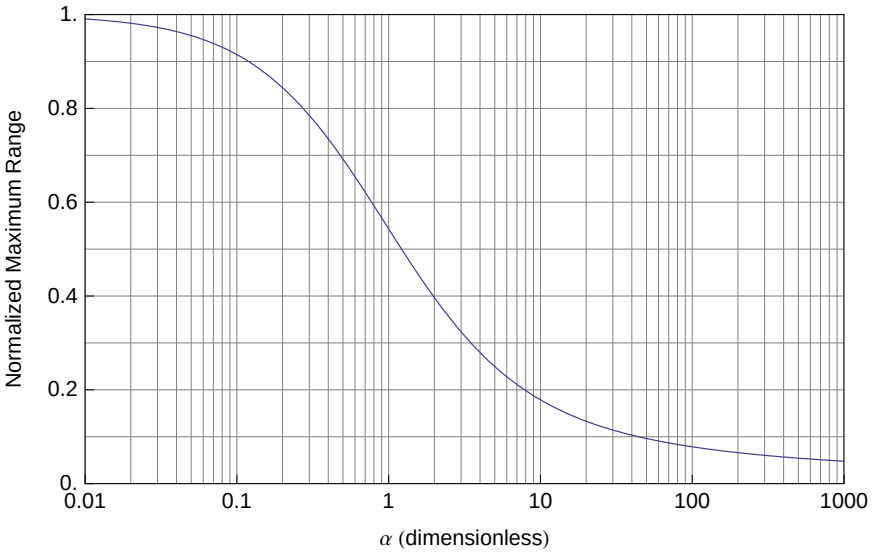

Figure 3. Normalized maximum range, $R_{\max}/R_0$, as a function of the dimensionless parameter $\alpha = bv_0/g$

So, while 45° is the optimal launch angle when there is no air resistance, this is clearly not the case when air resistance is taken into account. Indeed, we see that the optimal angle is a rapidly decreasing function of α.

3.2 Trajectory Height

The maximum height of the trajectory occurs when w is momentarily zero, which, from eq. (18), occurs when

$$t_{y=H} = \frac{1}{b}\ln(1+\alpha\sin\theta). \tag{39}$$

Substituting this time into eq. (19), we get the trajectory height

$$H = \frac{v_0\sin\theta}{b}\left[1 - \frac{\ln(1+\alpha\sin\theta)}{\alpha\sin\theta}\right] \tag{40}$$

and, from eq. (17), is located at the downrange distance

$$x_{y=H} = \frac{v_0\cos\theta}{b}\left(\frac{\alpha\sin\theta}{1+\alpha\sin\theta}\right). \tag{41}$$

3.3 Total Time of Flight and Average Speed

Using eqs. (17), (21), and (28), the total time of flight is

$$T = \frac{1}{b}\left[1+\alpha\sin\theta+W(z)\right], \tag{42}$$

where

$$z = -(1+\alpha\sin\theta)\exp[-(1+\alpha\sin\theta)] \quad \text{and} \quad \alpha \equiv bv_0/g.$$

The average speed over the entire flight is $\bar{v} = R/T$, which gives

$$\bar{v} = \frac{v_0\cos\theta}{1+\alpha\sin\theta}. \tag{43}$$

3.4 Impact Velocity and Impact Angle

Impact occurs when $x = R$. The components of the impact velocity at this point are

$$u = -u_0\frac{W(z)}{1+\alpha\sin\theta} \quad \text{and} \quad w = -w_0\frac{1+W(z)}{\alpha\sin\theta}, \tag{44}$$

and the impact angle is

$$\phi = \tan^{-1}\left[\left(\frac{1+\alpha\sin\theta}{\alpha\sin\theta}\right)\left(\frac{1+W(z)}{W(z)}\right)\tan\theta\right], \tag{45}$$

where θ is the launch angle, and again,

$$z = -(1+\alpha\sin\theta)\exp[-(1+\alpha\sin\theta)] \quad \text{and} \quad \alpha \equiv bv_0/g.$$

An interesting property of the maximum-range trajectory is that *the impact velocity is perpendicular to the launch velocity*. [11] We can show this by demonstrating that the dot product of the two vectors vanishes. Using eq. (44) and simplifying,

$$\boldsymbol{v}_0 \cdot \boldsymbol{v} = (u_0\hat{\boldsymbol{\imath}} + w_0\hat{\boldsymbol{\jmath}}) \cdot \left(-u_0 \frac{W}{1+\alpha\sin\theta}\hat{\boldsymbol{\imath}} - w_0 \frac{1+W}{\alpha\sin\theta}\hat{\boldsymbol{\jmath}} \right) = -\frac{v_0^2}{1+\alpha\sin\theta}\left[\sin^2\theta + \frac{1+W}{\alpha}\sin\theta + W \right],$$

and from eq. (32) the term in brackets equals zero when $\theta = \theta_{\max}$. Thus, the impact angle in this case is given by $\phi_{\max} = \theta_{\max} - \pi/2$.

The velocity components, u and w, are monotonically-decreasing functions of both α and θ, although u decreases more rapidly than w. Consequently, as long as $\alpha > 0$, the impact angle is always greater (*i.e.*, steeper) than the launch angle. The functional relationship is shown in Figures 4 and 5.

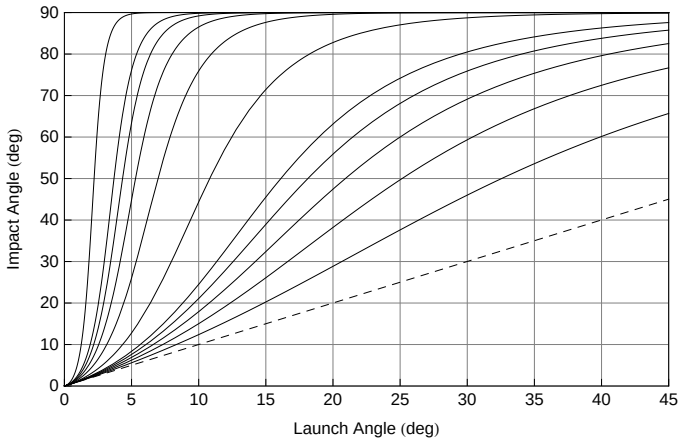

Figure 4. Impact angle, ϕ, as a function of launch angle, θ, for $\alpha = 1, 2, 3, 4, 5, 10, 20, 30, 40, 50, 100$. The dashed line is the limit as $\alpha \to 0$, which gives $\phi = \theta$

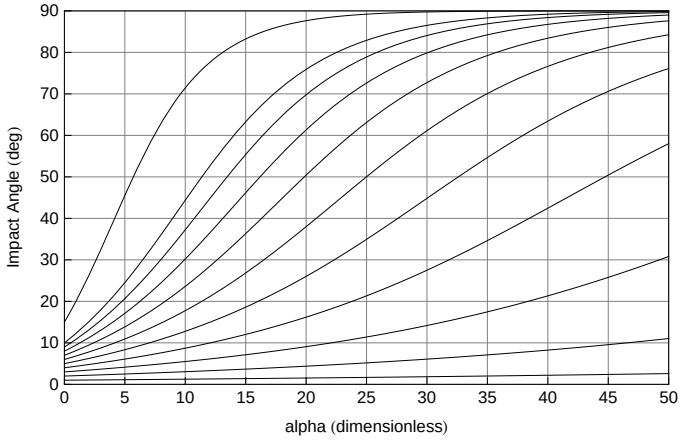

Figure 5. Impact angle as a function of α for $\theta = 0$ (lower curve), $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15$ degrees

The contour plot in Figure 6 shows the relationship between the launch angle, θ, and the dimensionless parameter α in order to achieve a given impact angle, ϕ. For example, if we want to achieve an impact angle of $80°$ or greater and $\alpha = 20$, then we need a launch angle $\theta \geq 11°$, but if $\alpha = 30$, then $\theta \geq 8°$.

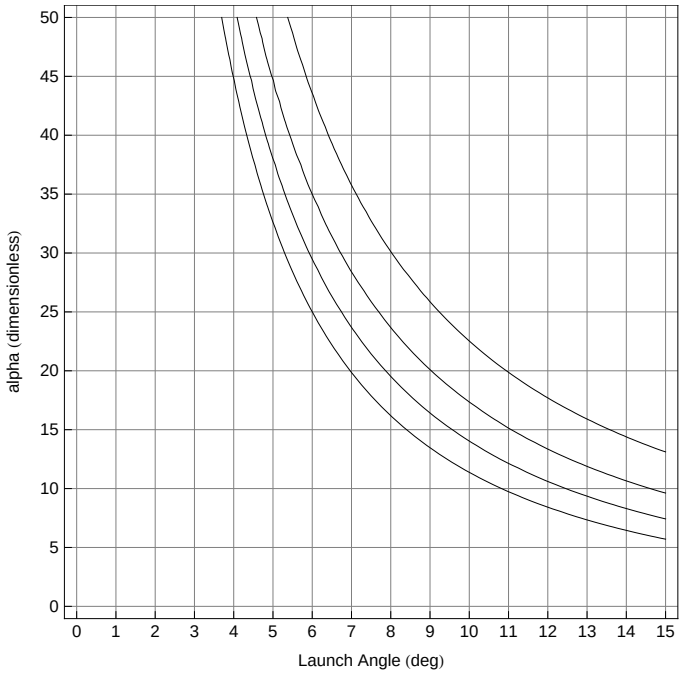

Figure 6. Contours of launch angle and α for $\phi = 50$ (lower curve), $60, 70, 80$ degrees

4 Projectile Motion with Quadratic Air Drag ($C_d \sim$ Constant)

We return to the quadratic drag force of eq. (13), and now we assume a constant drag coefficient, C_d, independent of the Mach number. The total force acting on the projectile then is

$$\boldsymbol{F} = -\tfrac{1}{2} C_d A \rho v^2 \hat{\boldsymbol{v}} - mg\hat{\boldsymbol{j}}, \tag{46}$$

and the equations of motion in terms of u and w are

$$\dot{u} = -bvu \quad \text{and} \quad \dot{w} = -bvw - g, \tag{47}$$

where

$$b \equiv \frac{\rho C_d A}{2m} \tag{48}$$

(which has dimensions of m^{-1} in SI units). Since $v = \sqrt{u^2 + w^2}$, the component equations are coupled, and the nature of this coupling prevents us from finding a general closed-form solution.[4] But if we limit ourselves to small launch angles, where $\tan\theta \ll 1$, then we can approximate $v = \sqrt{u^2 + w^2}$ with u, and this will permit a closed-form solution. This is the approximation that is analyzed throughout this secion.

4.1 Flat-Fire Trajectory Approximation

Flat-fire trajectory [8, 12] is the name given to launch angles a few degress above the horizontal, usually less than 10°. In this case the horizontal velocity is on average much greater than the vertical velocity, $u \gg w$. Therefore, $v = \sqrt{u^2 + w^2} \approx u$, and the equations of motion become

$$\dot{u} = -bu^2 \quad \text{and} \quad \dot{w} = -buw - g. \tag{49}$$

[4] It is not difficult to solve these equations numerically, and the standard technique using the fourth-order Runge-Kutta method is contained in the Appendix B.

The first equation is easily integrated to give the horizontal velocity

$$u = \frac{u_0}{1 + bu_0 t}. \tag{50}$$

Another integration gives x as a function of time,

$$x = \frac{1}{b} \ln(1 + bu_0 t), \tag{51}$$

where we are assuming the projectile is launched from the origin. To integrate the second equation of eq. (47), let's first use the chain rule,

$$\frac{dw}{du}\frac{du}{dt} = \frac{dw}{dt},$$

and eq. (47), to write

$$-bu^2 \frac{dw}{du} = -buw - g.$$

Then it is not difficult to rearrange this to

$$d\left(\frac{w}{u}\right) = -\frac{g}{2b} d\left(\frac{1}{u^2}\right),$$

from which the integration is now trivial, and solving for w gives

$$w = \frac{w_0 - gt/2}{1 + bu_0 t} - \frac{1}{2} gt. \tag{52}$$

Integrating again gives y as a function of time,

$$y = \left(w_0 + \frac{g}{2bu_0}\right) \frac{1}{bu_0} \ln(1 + bu_0 t) - \frac{gt}{2bu_0} - \frac{1}{4} gt^2. \tag{53}$$

Eliminating t between eqs. (49) and (51) gives the trajectory equation,

$$y = \frac{g}{(2bu_0)^2} \left[(1 + \beta \sin 2\theta) 2bx - (e^{2bx} - 1)\right], \tag{54}$$

where we have introduced the dimensionless parameter

$$\beta \equiv bv_0^2/g. \tag{55}$$

Once again, we see that accounting for air resistance results in a trajectory equation that involves x in a non-algebraic way. A plot of a typical trajectory is shown in Figure 7. Due to the effect of air resistance, the path is no longer a parabola, nor is it symmetrical about the maximum height.

The range, R, is the non-trivial value of x that gives $y = 0$, so that

$$(1 + \beta \sin 2\theta) 2bR - (e^{2bR} - 1) = 0. \tag{56}$$

By a now-familiar routine, we solve this equation for R by rearranging it into the Lambert W functional form. The steps are as follows. Multiply through by e^{-2bR} and rearrange to get

$$[(1 + \beta \sin 2\theta) 2bR + 1] e^{-2bR} = 1, \tag{57}$$

or

$$\left[-2bR - \frac{1}{1 + \beta \sin 2\theta}\right] e^{-2bR} = -\frac{1}{1 + \beta \sin 2\theta}.$$

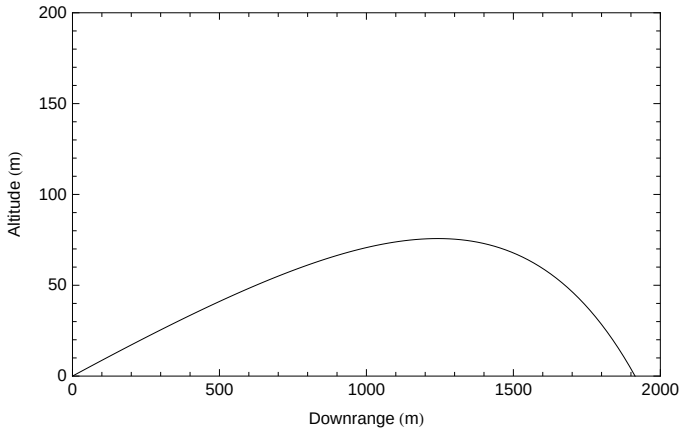

Figure 7. Flat-fire trajectory for a 5 degree firing angle and a 823 m/s muzzle velocity

Finally, multiply through by $\exp[-(1+\beta\sin 2\theta)^{-1}]$ to get

$$\left[-2bR - \frac{1}{1+\beta\sin 2\theta}\right]\exp\left(-2bR - \frac{1}{1+\beta\sin 2\theta}\right) = -\frac{1}{1+\beta\sin 2\theta}\exp\left(-\frac{1}{1+\beta\sin 2\theta}\right).$$

We recognize this equation to be of the form $W(z)e^{W(z)} = z$ with

$$z \equiv -(1+\beta\sin 2\theta)^{-1}\exp\left[-(1+\beta\sin 2\theta)^{-1}\right],$$

a known quantity, and

$$W(z) = -2bR - \frac{1}{1+\beta\sin 2\theta}.$$

Therefore, the range is given by

$$R = -\frac{1}{2b}\left[W(z) + \frac{1}{1+\beta\sin 2\theta}\right], \tag{58}$$

where $W(z)$ is the Lambert function[5],

$$z \equiv -(1+\beta\sin 2\theta)^{-1}\exp\left[-(1+\beta\sin 2\theta)^{-1}\right] \quad \text{and} \quad \beta \equiv bv_0^2/g. \tag{59}$$

Notice that the firing angle θ enters into the range formula only through the particular combination $\beta\sin 2\theta$.[6] Furthermore, notice that the firing angle dependence only involves the combination of $\sin 2\theta$, the same as the case of no air resistance. This would imply that the optimum angle is again 45°. But this has no practical significance since we must restrict the launch angle to be small for the approximation to be valid in the first place!

[5] Since R must be positive, we require that $W(z) < -(1+\beta\sin 2\theta)^{-1}$ in eq. (56), which in the limit as $\beta \to 0$ requires $W(z) < -1$. This implies that here we want $W_{-1}(z)$, the secondary branch of the Lambert W function, which varies from -1 to $-\infty$ as z varies from $-1/e$ to 0 (see Appendix A). In Mathematica®, the principal branch is known as `LambertW[z]` and the secondary branch is known as `LambertW[-1,z]`. C++ code for both branches is listed in Appendix B.

[6] Contrast this with the case of linear air resistance, eq. (28), where the firing angle enters into the range formula outside of α.

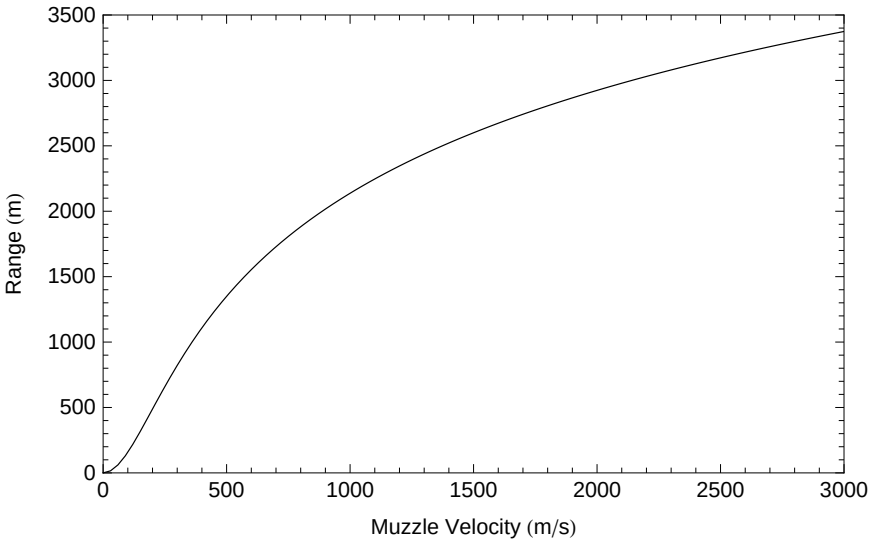

Figure 8. Range as a function of muzzle velocity for a 5 degree firing angle

4.2 Trajectory Height

The maximum height of the trajectory, H, can be obtained by setting $dy/dx = 0$, solving for x, and substituting back into the formula for y. This gives

$$H = \frac{\tan\theta}{2b}\left[\frac{1+\beta\sin 2\theta}{\beta\sin 2\theta}\ln(1+\beta\sin 2\theta) - 1\right] \tag{60}$$

and occurs when

$$t_{y=H} = \frac{\sqrt{1+\beta\sin\theta} - 1}{bv_0\cos\theta}.$$

The maximum height is located at the downrange distance

$$x_{y=H} = \frac{1}{2b}\ln(1+\beta\sin 2\theta).$$

4.3 Total Time of Flight and Average Speed

The total time of flight, T, is

$$T = \frac{\sqrt{-W(z)(1+\beta\sin 2\theta)} - 1}{bv_0\cos\theta}, \tag{61}$$

where we used eqs. (49), (55) and (58).

The average speed over the entire flight is $\bar{v} = R/T$, and we find

$$\bar{v} = v_0\cos\theta\frac{\sqrt{-W(z)(1+\beta\sin 2\theta)} + 1}{2(1+\beta\sin 2\theta)}. \tag{62}$$

4.4 Impact Velocity and Impact Angle

Let us also calculate the velocity of the projectile as a function of its distance downrange. From the chain rule,

$$\frac{du}{dx} = \frac{\dot{u}}{u} \quad \text{and} \quad \frac{dw}{dx} = \frac{\dot{w}}{u}. \tag{63}$$

Using eq. (47), we get

$$\frac{du}{dx} = -bu \quad \text{and} \quad \frac{dw}{dx} = -bw - \frac{g}{u}.$$

The first of these is easily integrated,

$$u = u_0 e^{-bx}, \tag{64}$$

and when substituted into the second, we get the differential equation

$$\frac{dw}{dx} + bw = -\frac{g}{u_0} e^{bx}.$$

Multiplying through by e^{bx} gives

$$e^{bx}\frac{dw}{dx} + be^{bx}w = \frac{d}{dx}(we^{bx}) = -\frac{g}{u_0}e^{2bx},$$

which is easily integrated to give

$$w = w_0 e^{-bx} - \frac{g}{2bu_0}(e^{bx} - e^{-bx}). \tag{65}$$

At impact, $x = R$, and we get

$$u = \frac{v_0 \cos\theta}{\sqrt{-W(z)(1+\beta\sin 2\theta)}} \tag{66}$$

for the horizontal component and

$$w = \frac{v_0 \sin\theta}{\sqrt{-W(z)(1+\beta\sin 2\theta)}} \left[1 + \frac{W(z)(1+\beta\sin 2\theta)+1}{\beta \sin 2\theta}\right] \tag{67}$$

for the vertical component of the impact velocity. The impact angle, ϕ, is given by

$$\phi = \tan^{-1}\left[\left(1 + \frac{W(z)(1+\beta\sin 2\theta)+1}{\beta\sin 2\theta}\right)\tan\theta\right], \tag{68}$$

where

$$z = -(1+\beta\sin 2\theta)^{-1}\exp\left[-(1+\beta\sin 2\theta)^{-1}\right] \quad \text{and} \quad \beta = bv_0^2/g.$$

The impact angle is plotted in Figure 9 as a function of the launch angle for various values of β. As in the case of linear air resistance, the impact angle is always greater (steeper) than the launch angle as long as $\beta > 0$. However, we see that in the case of quadratic resistance, the effect is not quite so pronounced as that of linear air resistance (*cf.* Figure 4).

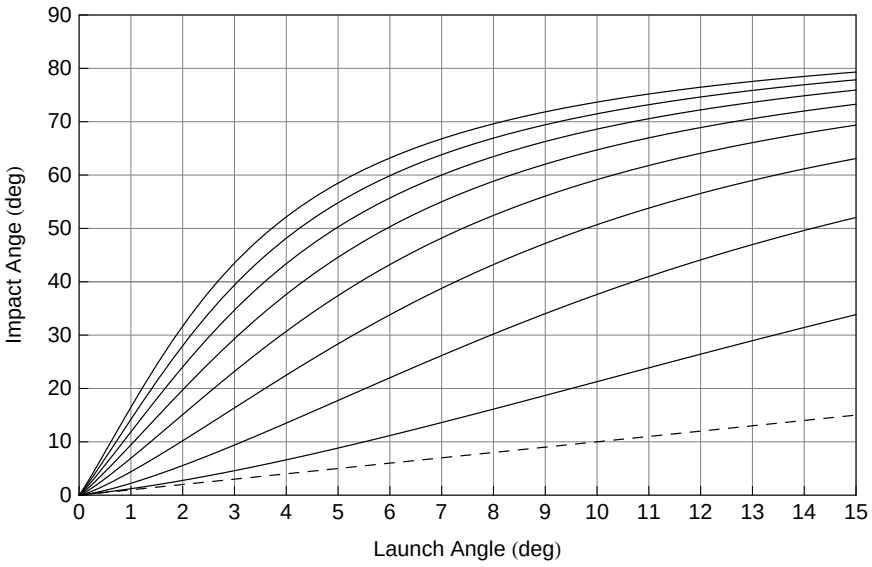

Figure 9. Impact angle as a function of θ for $\beta = 0$ (dashed), $10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7, 10^8$

5 Discussion

In general, the drag coefficient is a complex function of both the Reynold's number, Re, and the Mach number, Ma [8, 13] In the case of exterior ballistics through the air that we are considering here, $\text{Re} > 10^5$ and in this regime, the drag coefficient tends to be insensitive to the Reynold's number and becomes a function of Mach number only. [14, 15] Increasing the speed through the speed of sound (Mach I) results in a steep increase in the drag coefficient, it reaches a peak value slightly beyond Mach I, and then it gradually decreases in value as the speed continues to increase.

The drag coefficient for cubes and spheres have been studied extensively and drag curves have been fitted, [16, 17] as shown in Figure 10. These curve fits are as follows:

$$C_d(\text{Ma})_{\text{sphere}} = \begin{cases} 0.45\text{Ma}^2 + 0.424 & \text{if } 0 \leq \text{Ma} \leq 0.722 \\ 2.1e^{-1.2(\text{Ma}+0.35)} - 8.9e^{-2.2(\text{Ma}+0.35)} + 0.92 & \text{if Ma} > 0.722 \end{cases} \tag{69}$$

$$C_d(\text{Ma})_{\text{cube}} = \begin{cases} 0.60\text{Ma}^2 + 1.04 & \text{if } 0 \leq \text{Ma} \leq 1.131 \\ 2.1e^{-1.16(\text{Ma}+0.35)} - 6.5e^{-2.23(\text{Ma}+0.35)} + 1.67 & \text{if Ma} > 1.131 \end{cases} \tag{70}$$

Drag curves for bullets depend upon the nose and shape and show greater variation than the curves in Figure 10 [8].

5.1 Measuring the Drag Coefficient

We saw that in the case of linear air resistance, the horizontal component of the velocity falls off linearly with distance downrange: $u = u_0 - bx$. Now, if we restrict ourselves to flat-fire trajectories, then we can neglect w as compared to u, and $v \approx u$. Then it follows from from eqs. (16), (17), (48), and (49) that

$$v = v_0 - bx \quad \text{and} \quad \ln v = \ln v_0 - bt \tag{71}$$

if and only if $C_d \sim 1/\text{Ma}$, and

$$\ln v = \ln v_0 - bx \quad \text{and} \quad \frac{1}{v} = \frac{1}{v_0} + bt \tag{72}$$

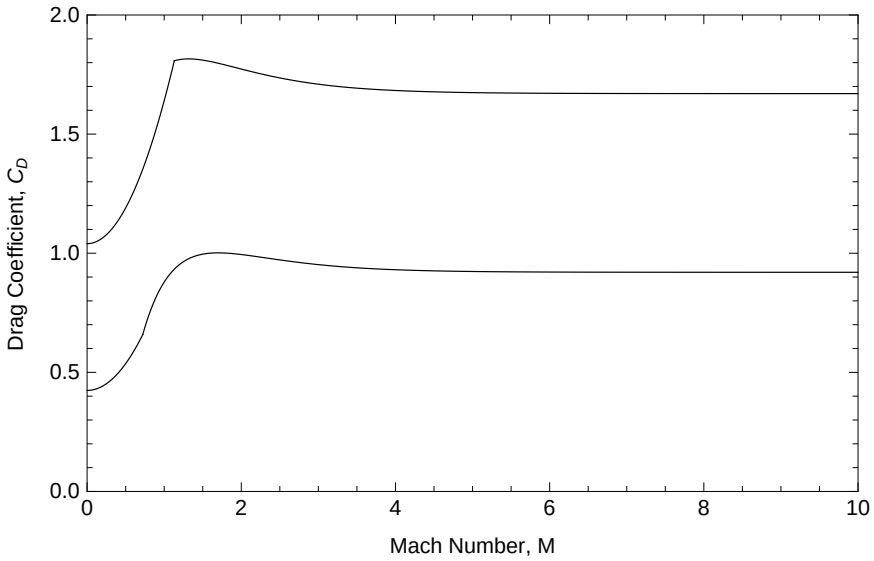

Figure 10. Drag coefficient as a function of Mach number for cube (upper curve) and sphere (lower)

if and only if $C_d \sim$ Constant. Thus, if we have velocity break screens positioned at various locations downrange and record the velocity at each one, along with the downrange distance x and the time of impact t, then these equations tell us how to find the dimensionless coefficient b from the slope of the straight line. Once we have determined b, we can calculate C_d from eqs. (14) and (46).

A more systematic approach for treating a general drag curve is contained in the reports by Weinacht, Cooper and Newill. [18, 19]

5.2 Sample Case

Let's consider a sample case of a 1-kg steel sphere with a constant drag coefficient of $C_d = 0.5$ and a launch speed of 1000 m/s. In the case of linear air resistance, we find that the angle for maximum range is $\theta_{\max} = 8°$ and $R_{\max} = 3111$ m. This is shown in Figure 5.2, where it is clear that 45° no longer gives the maximum range. Also notice how quickly the impact angle approaches 90° as the launch angle increases.

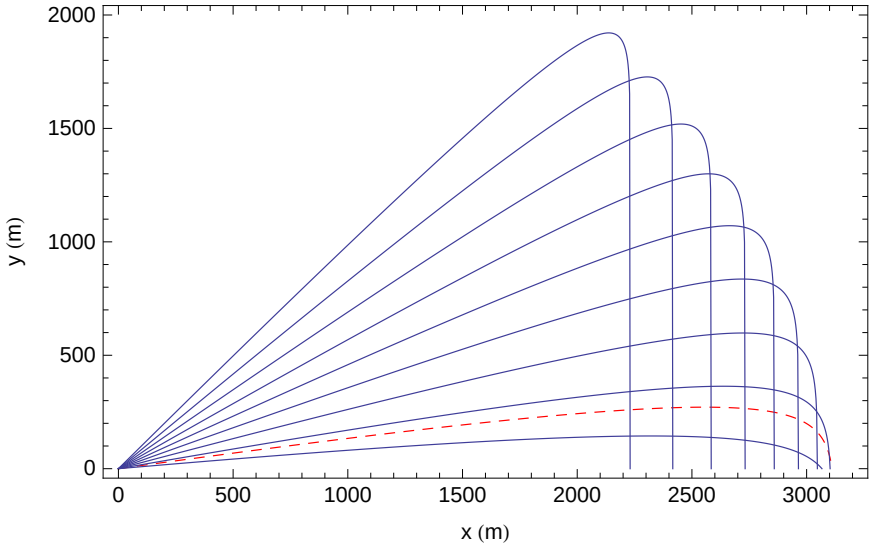

Figure 11. Linear air drag trajectories for $\theta = 5°$, $10°$, $15°$, $20°$, $25°$, $30°$, $35°$, $40°$, $45°$ (solid blue) and $\theta_{\text{max}} = 8°$ (dashed red)

The value of b for this example can be calculated from eq. (14) and we find $b = 0.317\text{s}^{-1}$, which means that the air drag deceleration is initially $bv_0 = 317$ m/s^2, or about 32 G's.

In the case of quadratic air resistance, we could only solve the equations exactly when we made the approximation of shallow launch angles (*flat-fire approximation*). The resulting equations lead to $\theta = 45°$ as the optimal angle, but this would invalidate the approximation that $\tan\theta \ll 1$. Under these circumstances, we are not in a position to optimize the range.

However, we can ask how well our flat-fire approximate trajectories compare to those from the Runge-Kutta solution of the exact quadratic air drag problem. This is shown in Figure 5.2. The approximate closed-form solution overpredicts the range by only 0.2% at 5° and by less than 1% at 10°. Even at 15°, it overpredicts the range by only 2%. The value of b for this example can be calculated from eq. (46) and we get $b = 9.24 \times 10^4$ m^{-1}, which means that the air drag deceleration is initially $bv_0^2 = 924$ m/s^2, or about 94 G's.

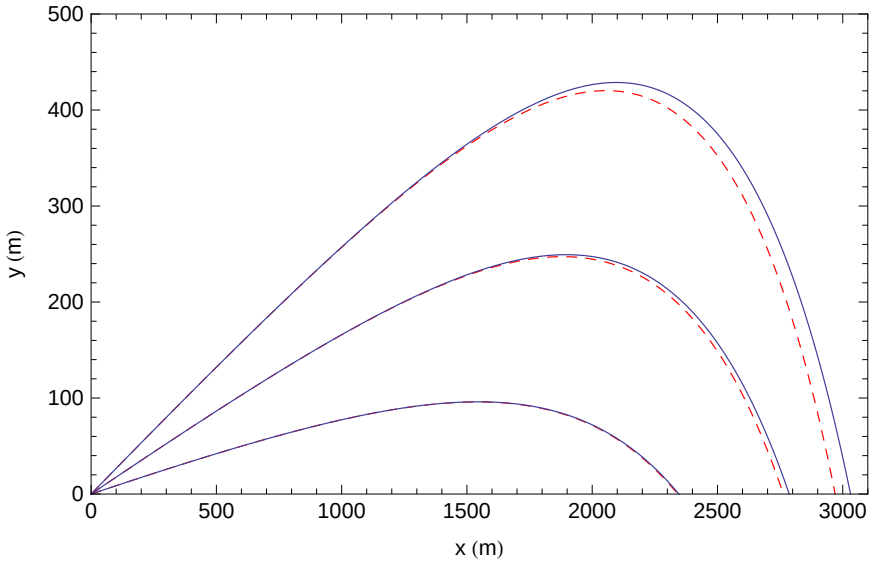

Figure 12. Quadratic air drag trajectories in the flat-fire approximation (solid blue) compared to Runge-Kutta solutions (dashed red) for $\theta = 5°$, $10°$, $15°$

To find the launch angle that will result in the maximum range, we made a number of Runge-Kutta runs. Figure 13. shows that in this case the maximum range is obtained when the launch angle is 25°.

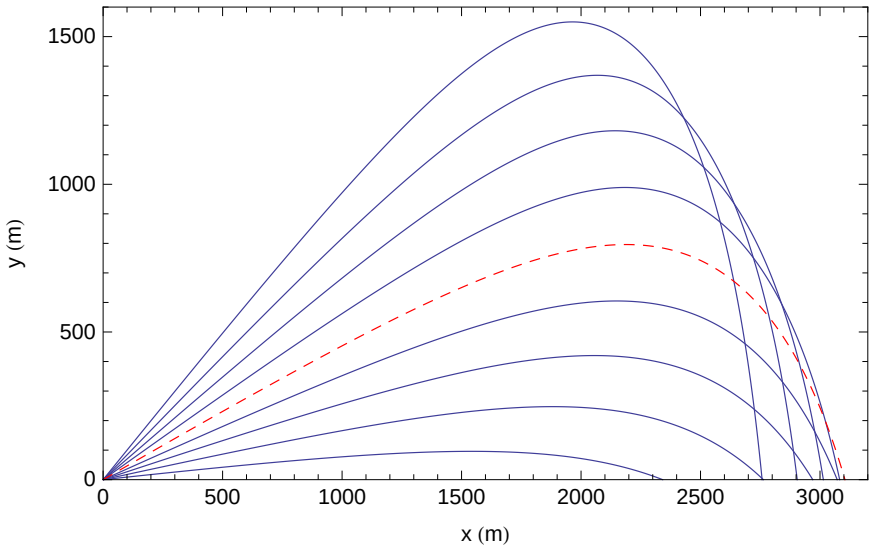

Figure 13. Quadratic air drag trajectories using Runge-Kutta for $\theta = 5°, 10°, \cdots, 45°$ (solid blue) with maximum range at $25°$ (dashed red)

References

[1] Walters, W. P. and Segletes, S. B., "Distance Traveled by a Hypervelocity Projectile in Air," ARL-TR-5612, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, July 2011.

[2] Segletes, S. B. and Walters, W. P., "Analytical Solution in Curvilinear Coordinates for the Trajectory of a Projectile Subject to Aerodynamic Drag," ARL-TR-5822, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, December 2011.

[3] O'Malley, M., Parker, R., Bradley, G. and Stancoff, C., "Calculation and Visualization of Air-Drag and Gravity Effects for Modeling Fragment Flight Paths in MUVES-S2," ARL-TR-5782, September 2011.

[4] R. M. Corless, G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey and D. E. Knuth, "On the LambertW function," *Advances in Computational Mathematics*, 1996, Volume 5, Number 1, pp. 329-359. Also available online at https://www.cs.uwaterloo.ca/research/tr/1993/03/W.pdf.

[5] Veberič, D., "Having Fun with Lambert W(x) Function," arXiv:1003.1628v1 [cs.MS] 8 Mar 2010.

[6] Warburton, R. D. H. and Wang, J., "Analysis of asymptotic projectile motion with air resistance using the Lambert W function," Am. J. Phys., **72**, (11), November 2004, pp. 1404-1407.

[7] Warburton, R. D. H., Wang, J., and Burgdorfer, J., "Analytic Approximations of Projectile Motion with Quadratic Air Resistance," J. Service Science & Management, **3**, 2010, pp. 98-105.

[8] McCoy, R. L., *Modern Exterior Ballistics: The Launch and Flight Dynamics of Symmetric Projectiles*, Schiffer Publishing, 1999.

[9] Celmins, I., "Projectile Supersonic Drag Characteristics," Ballistic Research Laboratory, Memorandum Report BRL-MR-3843, July, 1990.

[10] Stewart, S. M., "On the trajectories of projectiles depicted in early ballistic woodcuts," *Eur. J. Phys.*, **33**, 2012, pp. 149-166.

[11] Morales, D. A., "A generalization on projectile motion with linear resistance," Can. J. Phys., **89**, 2011, pp. 1233-1250.

[12] Carlucci, D. E. and Jacobson, S. S., *Ballistics: Theory and Design of Guns and Ammunition*, CRC Press, 2007

[13] Farrar, C. L. and Leeming, D. W., *Military Ballistics: A Basic Manual*, Pergamon Press, 1983.

[14] Bailey, A. B. and Hiatt, J., *Free-Flight Measurements of Sphere Drag at Subsonic, Transonic, Supersonic, and Hypersonic Speeds for Continuum, Transition, and Near-Free-Molecular Flow Conditions*, AEDC-TR-70-291, Air Force Cambridge Research Laboratories, Bedford, MA, March 1971.

[15] Klimi, G., *Exterior Ballistics with Applications: Skydiving, Parachute Fall, Flying Fragments*, Xlibris Corp., 2008.

[16] Carter, R. T., Jandir, P. S. and Kress, M. E., "Estimating the Drag Coefficients of Meteorites for All Mach Number Regimes," *40th Lunar and Planetary Science Conference*," 2009.

[17] Carter, R. T., Jandir, P. S. and Kress, M. E., "Constraining the Drag Coefficients of Meteors in Dark Flight," NASA, Huntsville, AL July 2011.

[18] Weinacht, P., Cooper, G. R. and Newill, J. F., "Analytical Prediction of Trajectories for High-Velocity Direct-Fire Munitions," ARL-TR-3567, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, August 2005.

[19] Cooper, G. R., Weinacht, P. and Newill, J. F., "Another Analytical Approach to Predicting Munition Trajectories," ARL-TR-3948, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD, September 2006.

[20] Press, W. H., Teukosky, S. A., Vetterling, W. T., and Flannery, B. P., *Numerical Recipes in C, Second Edition*, Cambridge University Press, 1992.

Appendices

Appendix A Plot of the Lambert W Function

First consider the equation $y = xe^x$, which is shown plotted in Figure 14. It is defined for $-\infty < x < +\infty$ and has an absolute minimum of $y = -1/e$ at $x = -1$. As $x \to -\infty$, $y \to 0^-$, and as $x \to +\infty$, $y \to +\infty$. Substituting a value for x, we can readily calculate a value for y. Now consider the inverse, $ye^y = x$. For

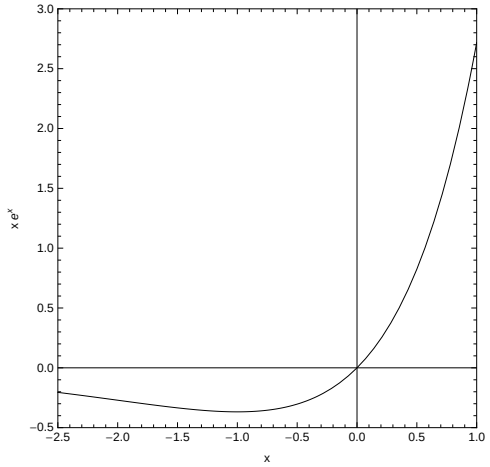

Figure 14. Plot of $y = xe^x$

a given value of x, the solution to this equation is $y = W(x)$, where $W(x)$ is the Lambert function, and is shown plotted in Figure 15.

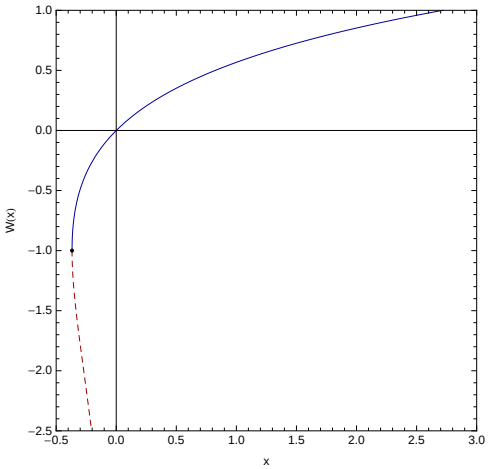

Figure 15. Plot of $W(x)$

There are two branches to the function: $W_0(x)$, which is shown in solid blue, is defined on the half-open interval $[-1/e, +\infty)$, where $W_0(x)$ varies from -1 to $+\infty$; and $W_{-1}(x)$, which is shown in dashed red, is defined on the half-open interval $[-1/e, 0)$, where $W_{-1}(x)$ varies from -1 to $-\infty$.

Appendix B C++ Source Code Listings

The C++ source code listing for both branches of the Lambert W function: [5]

Listing 1. lambertW.h

```
// lambertW.h: header file for both branches of the Lambert W function
// Ref: "Having Fun with Lambert W(x) Function," Darko Veberic, arXiv:1003.1628v1 [cs.MS], 8 Mar 2010
// R. Saucier, July 2012

#ifndef LAMBERT_H
#define LAMBERT_H

#include <iostream>
#include <cstdlib>
#include <cmath>
using namespace std;

double lambertW_0( double z ) {   // principal branch; domain is half-open interval [-1/e, infinity) and range is half-open interval [-1, infinity)

    const double P[10] = { -1., 1., -1./3., 11./72., -43./540., 769./17280., -221./8505., -680863./43545600., -1963./204120., 226287557./37623398400. };
    const double A[5]  = { 1., 5.931375839364438, 11.39220550532913, 7.33888339911111, 0.653449016991959 };
    const double B[5]  = { 1., 6.931373689597704, 16.82349461388016, 16.43072324143226, 5.115235195211697 };
    const double C[5]  = { 1., 2.445053070726557, 1.343664225958226, 0.148440055397592, 0.0008047501729130 };
    const double D[5]  = { 1., 3.444708986486002, 3.292489857371952, 0.916460018803122, 0.0530686404483322 };

    double p, a, a2, a3, a4, a5, b, b2, b3, b4;
    if ( -1. / M_E <= z && z < -0.32358170806015724 ) {
       p = sqrt( 2. * ( 1. + M_E * z ) );
       return P[0] + p * ( P[1] + p * ( P[2] + p * ( P[3] + p * ( P[4] + p * ( P[5] + p * ( P[6] + p * ( P[7] + p * ( P[8] + p * ( P[9] ) ) ) ) ) )
              ) ) );
    }
    else if ( -0.32358170806015724 <= z && z < 0.14546954290661823 ) {
       return z * ( A[0] + z * ( A[1] + z * ( A[2] + z * ( A[3] + z * A[4] ) ) ) / ( B[0] + z * ( B[1] + z * ( B[2] + z * ( B[3] + z * B[4] ) )
              ) );
    }
    else if ( 0.14546954290661823 <= z && z < 8.706658967856612 ) {
       return z * ( C[0] + z * ( C[1] + z * ( C[2] + z * ( C[3] + z * C[4] ) ) ) / ( D[0] + z * ( D[1] + z * ( D[2] + z * ( D[3] + z * D[4] ) )
              ) );
    }
    else if ( 8.706658967856612 <= z ) {
       a = log( z );
       a2 = a * a;
       a3 = a * a2;
       a4 = a * a3;
       a5 = a * a4;
       b = log( log( z ) );
       b2 = b * b;
       b3 = b * b2;
       b4 = b * b3;
       return a - b + b / a + ( -2. + b ) * b / ( 2. * a2 ) + ( 6. - 9. * b + 2. * b2 ) * b / ( 6. * a3 ) +
              ( -12. + 36. * b - 22. * b2 + 3. * b3 ) * b / ( 12. * a4 ) +
              ( 60. - 300. * b + 350. * b2 - 125. * b3 + 12. * b4 ) * b / ( 60. * a5 );
    }
    else {
       cerr << "Error: z = " << z << " outside domain of lambertW_0(z)" << endl << "Program stopped" << endl;
       exit( EXIT_FAILURE );
    }
}

double lambertW_1( double z ) {   // secondary branch; domain is half-open interval [-1/e, 0) and range is half-open interval (-infinity, -1]

    const double P[10] = { -1., 1., -1./3., 11./72., -43./540., 769./17280., -221./8505., -680863./43545600., -1963./204120., 226287557./37623398400. };
    const double A[3]  = { -7.81417672390744, 253.88810188892484, 657.9493176902304 };
    const double B[6]  = { 1., -60.43958713690808, 99.9856708310761, 682.6073999909428, 962.1784396969866, 1477.9341280760887 };

    double p, a, a2, a3, a4, a5, b, b2, b3, b4;
    if ( -1. / M_E <= z && z < -0.30298541769 ) {
       p = -sqrt( 2. * ( 1. + M_E * z ) );
       return P[0] + p * ( P[1] + p * ( P[2] + p * ( P[3] + p * ( P[4] + p * ( P[5] + p * ( P[6] + p * ( P[7] + p * ( P[8] + p * ( P[9] ) ) ) ) ) )
              ) ) );
    }
    else if ( -0.30298541769 <= z && z < -0.051012917658221676 ) {
       return ( A[0] + z * ( A[1] + z * ( A[2] ) ) ) / ( B[0] + z * ( B[1] + z * ( B[2] + z * ( B[3] + z * ( B[4] + z * ( B[5] ) ) ) ) ) );
    }
    else if ( -0.051012917658221676 < z && z < 0 ) {
       a = log( -z );
       a2 = a * a;
       a3 = a * a2;
       a4 = a * a3;
       a5 = a * a4;
       b = log( -log( -z ) );
       b2 = b * b;
       b3 = b * b2;
       b4 = b * b3;
       return a - b + b / a + ( -2. + b ) * b / ( 2. * a2 ) + ( 6. - 9. * b + 2. * b2 ) * b / ( 6. * a3 ) +
              ( -12. + 36. * b - 22. * b2 + 3. * b3 ) * b / ( 12. * a4 ) +
              ( 60. - 300. * b + 350. * b2 - 125. * b3 + 12. * b4 ) * b / ( 60. * a5 );
    }
    else {
       cerr << "Error: z = " << z << " outside domain of lambertW_1(z)" << endl << "Program stopped" << endl;
       exit( EXIT_FAILURE );
    }
}
#endif
```

Simple main program that makes use of the function:

Listing 2. lambertW.cpp

```cpp
// lambertW.C: Fast evaluation of both branches of the Lambert W function over its full domain [-1/e,infinity)
// Ref: "Having Fun with Lambert W(x) Function," Darko Veberic, arXiv:1003.1628v1 [cs.MS], 8 Mar 2010
// R. Saucier, July 2012

#include "lambertW.h"
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <cmath>
using namespace std;

int main ( int argc, char* argv[] ) {

   const double Z_MIN = -1. / M_E;

   double z = Z_MIN;
   if ( argc == 2 ) z = atof( argv[1] );

   if ( Z_MIN <= z && z < 0. ) {   // both branches are define in this domain
      cout << "z = " << z << " lambertW_0( z ) = " << lambertW_0( z ) << endl;
      cout << "z = " << z << " lambertW_1( z ) = " << lambertW_1( z ) << endl;
   }
   else if ( 0. <= z ) {   // only principal branch is defined in this domain
      cout << "z = " << z << " lambertW_0( z ) = " << lambertW_0( z ) << endl;
   }
   else {   // outside domain of both branches
     cerr << "Outside domain of function; program stopped" << endl;
     exit( EXIT_FAILURE );
   }

        return EXIT_SUCCESS;
}
```

Sample evaluations:

```
./lambertW
z = -0.367879 lambertW_0( z ) = -1
z = -0.367879 lambertW_1( z ) = -1

./lambertW -0.15
z = -0.15 lambertW_0( z ) = -0.179491
z = -0.15 lambertW_1( z ) = -2.99359

./lambertW 1.7
z = 1.7 lambertW_0( z ) = 0.779601
```

Implementation of the linear air resistance formulas contained in Section 3.

Listing 3. linear.cpp

```cpp
// linear.C: Projectile motion with air resistance linearly dependent on velocity (SI Units)
// R. Saucier, July 2012

#include "lambertW.h"
#include <fstream>
#include <iostream>
#include <cmath>
#include <cstdlib>
using namespace std;

double y( double G, double B, double v0, double th, double x  ) {   // trajectory: y as a function of x

   double u0 = v0 * cos( th );
   double w0 = v0 * sin( th );
   double z  = B * x / u0;
   return ( G / ( B * B ) ) * ( ( 1. + B * w0 / G ) * z + log( 1. - z ) );
}

double th_max( double G, double B, double v0  ) {   // launch angle for max range (rad)

   double alpha = B * v0 / G;
   if ( alpha == 1. ) return asin( 1. / ( M_E - 1. ) );
   double a = ( alpha - 1. ) * ( alpha + 1. );
   double W = lambertW_0( a / M_E );
   return asin( alpha * W / ( a - W ) );
}

double range_max( double G, double B, double v0  ) {   // max range over all launch angles (m)

   double c = v0 * v0 / G;
   double alpha = B * v0 / G;
   if ( alpha == 1. ) return c * sqrt( ( M_E - 2. ) / M_E );
   double a = ( alpha - 1. ) * ( alpha + 1. );
   double W = lambertW_0( a / M_E );
   return c * sqrt( ( alpha - ( 1. + W ) ) * ( alpha + ( 1. + W ) ) / ( alpha * alpha * a ) );
}

double range_norm( double alpha ){   // normalized max range (dimensionless)

   if ( alpha == 1. ) return ( M_E - 1. ) * ( M_E - 1. ) / ( 2. * M_E );
   double a = ( alpha - 1. ) * ( alpha + 1. );
   double W = lambertW_0( a / M_E );
   return ( a - W ) * ( a - W ) / ( 2. * alpha * alpha * a * W );
}

double range( double G, double B, double v0, double th ) {   // range at the given launch angle (m)

   double u0    = v0 * cos( th );
```

```
49      double alpha = B * v0 / G;
50      double a     = 1. + alpha * sin( th );
51      double z     = -a * exp( -a );
52      return ( u0 / B ) * ( 1. + lambertW_0( z ) / a );
53  }
54
55  double height( double G, double B, double v0, double th ) {   // max y over the trajectory (m)
56
57       double w0    = v0 * sin( th );
58       double alpha = B * v0 / G;
59       double a     = alpha * sin( th );
60       return ( w0 / B ) * ( 1. - log( 1. + a ) / a );
61  }
62
63  double t_height( double G, double B, double v0, double th  ) {   // time to reach max height of trajectory (m)
64
65      double alpha = B * v0 / G;
66      return log( 1. + alpha * sin( th ) ) / B;
67  }
68
69  double x_height( double G, double B, double v0, double th  ) {   // x distance when at max height (m)
70
71      double u0    = v0 * cos( th );
72      double alpha = B * v0 / G;
73      double a     = alpha * sin( th );
74      return ( u0 / B ) * a / ( 1. + a );
75  }
76
77  double t_range( double G, double B, double v0, double th  ) {   // total time of flight (s)
78
79      double alpha = B * v0 / G;
80      double a    = 1. + alpha * sin( th );
81      double z = -a * exp( -a );
82      return  ( a + lambertW_0( z ) ) / B;
83  }
84
85  double v_bar( double G, double B, double v0, double th  ) {   // average speed over entire flight (s)
86
87      double u0    = v0 * cos( th );
88      double alpha = B * v0 / G;
89      double a     = 1. + alpha * sin( th );
90      return  u0 / a;
91  }
92
93  double u_impact( double G, double B, double v0, double th ) {   // x-component of velocity at impact (m/s)
94
95      double u0    = v0 * cos( th );
96      double alpha = B * v0 / G;
97      double a     = 1. + alpha * sin( th );
98      double z     = -a * exp( -a );
99      double W     = lambertW_0( z );
100     return -u0 * W / a;
101 }
102
103 double w_impact( double G, double B, double v0, double th ) {   // y-component of velocity at impact (m/s)
104
105     double w0    = v0 * sin( th );
106     double alpha = B * v0 / G;
107     double a     = 1. + alpha * sin( th );
108     double z     = -a * exp( -a );
109     double W     = lambertW_0( z );
110     return -w0 * ( 1. + W ) / ( a - 1. );
111 }
112
113 double v_impact( double G, double B, double v0, double th ) {   // speed at impact (m/s)
114
115   double u = u_impact( G, B, v0, th );
116   double w = w_impact( G, B, v0, th );
117   return sqrt( u * u + w * w );
118 }
119
120 double th_impact( double G, double b, double v0, double th ) {   // angle at impact (rad)
121
122     double alpha = b * v0 / G;
123     double a     = 1. + alpha * sin( th );
124     double z     = -a * exp( -a );
125     double W     = lambertW_0( z );
126     return atan( ( a / ( a - 1. ) ) * ( ( 1. + W ) / W ) * tan( th ) );
127 }
128
129 double u( double B, double v0, double th, double t ) {   // x-component of velocity at time t (m/s)
130
131     return v0 * cos( th ) * exp( -B * t );
132 }
133
134 double w( double G, double B, double v0, double th, double t ) {   // y-component of velocity at time t (m/s)
135
136     return ( v0 * sin( th ) + G / B ) * exp( -B * t ) - G / B;
137 }
138
139 int main( int argc, char* argv[] ) {
140
141     const double D2R     = M_PI / 180.;                              // to convert degrees to radians
142     const double R2D     = 180. / M_PI;                              // to convert radians to degrees
143     const double G       = 9.80665;                                  // acceleration due to gravity (m/s^2)
144     const double RHO_AIR = 1.205;                                    // density of air at 20 deg Celsius (kg/m^3)
145     const double V_SOUND = 343.2;                                    // speed of sound in air at 20 deg Celsius (m/s)
146     const double m       = 1.;                                       // mass of projectile (kg)
147     const double rho     = 7830.;                                    // density of steel (kg/m^3)
148     const double D       = pow( 6. * m / ( M_PI * rho ), 1./3. );    // diameter of spherical projectile (m)
149     const double Ap      = 0.25 * M_PI * D * D;                      // presented area of spherical projectile (m^2)
150     const double Cd      = 0.5;                                      // drag coefficient (dimensionless)
151     const double B       = RHO_AIR * V_SOUND * Cd * Ap / ( 2. * m );   // lumped drag coefficient (s^(-1))
152
153     double v0 = 1000.;                      // launch speed (m/s)
154     double deg = th_max( G, B, v0 ) * R2D;    // default launch angle is for max range (deg)
155     if ( argc == 2 ) deg = atof( argv[1] );   // launch angle specified on command line (deg)
```

```
    double th = deg * D2R;                    // launch angle converted to rad (rad)

    double alpha = B * v0 / G;                // dimensionless ratio of deceleration due to air drag and acceleration due to gravity
    double R     = range( G, B, v0, th );     // range at given launch angle (m)

    cout << "Launch angle (deg)                    = " << deg                              << endl
         << "b (1/s)                               = " << B                                << endl
         << "alpha (-)                             = " << alpha                            << endl
         << "Optimal angle for max range (deg)     = " << th_max( G, B, v0 ) * R2D         << endl
         << "Max range (m)                         = " << range_max( G, B, v0 )            << endl
         << "Normalized max range, Rmax/R0 (-)     = " << range_norm( alpha )              << endl
         << "Range at given launch angle (m)       = " << R                                << endl
         << "Maximum height of trajectory (m)      = " << height( G, B, v0, th )           << endl
         << "Elapsed time to achieve max height (s) = " << t_height( G, B, v0, th )        << endl
         << "Downrange distance at max height (m)  = " << x_height( G, B, v0, th )         << endl
         << "Total time of flight (s)              = " << t_range( G, B, v0, th  )         << endl
         << "Average speed over entire flight (m/s) = " << v_bar( G, B, v0, th  )          << endl
         << "Impact speed (m/s)                    = " << v_impact( G, B, v0, th )         << endl
         << "Impact angle (deg)                    = " << th_impact( G, B, v0, th ) * R2D << endl;

    ofstream fout( "linear.out" );
    for ( double x = 0.; x <= R; x += R / 999. ) fout << x << "\t" << y( G, B, v0, th, x ) << endl;
    fout << R << "\t" << 0. << endl;
    fout.close();

    return EXIT_SUCCESS;
}
```

Implementation of the quadratic air resistance formulas contained in Section 4.

Listing 4. quadratic.cpp

```
// quadratic.C: Projectile motion with air resistance quadratically dependent on velocity (SI Units)
// R. Saucier, July 2012

#include "lambertW.h"
#include <fstream>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <cassert>
using namespace std;

inline double sqr( double x ) { return x * x; }

double y( double G, double B, double v0, double th, double x ) {   // trajectory: y as a function of x (m)

    double u0   = v0 * cos( th );
    double beta = B * v0 * v0 / G;
    double a    = 1. + beta * sin( 2. * th );
    double z    = 2. * B * x;
    return ( G / sqr( 2. * B * u0 ) ) * ( a * z - ( exp( z ) - 1. ) );
}

double range( double G, double B, double v0, double th ) {   // range (m)

    double beta = B * v0 * v0 / G;
    double zeta = 1. / ( 1. + beta * sin( 2. * th ) );
    double z    = -zeta * exp( -zeta );
    return -( lambertW_1( z ) + zeta ) / ( 2. * B );
}

double height( double G, double B, double v0, double th ) {   // height of trajectory (m)

    double beta = B * v0 * v0 / G;
    double z    = 1. + beta * sin( 2. * th );
    return ( tan( th ) / ( 2. * B ) ) * ( z * log( z ) / ( z - 1. ) - 1. );
}

double t_height( double G, double B, double v0, double th ) {   // time to reach max height of trajectory (m)

    double beta = B * v0 * v0 / G;
    return ( sqrt( 1. + beta * sin( 2. * th ) ) - 1. ) / ( B * v0 * cos( th ) );
}

double x_height( double G, double B, double v0, double th ) {   // downrange distance at max height of trajectory (m)

    double beta = B * v0 * v0 / G;
    double z    = 1. + beta * sin( 2. * th );
    return log( z ) / ( 2. * B );
}

double t_range( double G, double B, double v0, double th  ) {   // total time of flight

    double beta = B * v0 * v0 / G;
    double zeta = 1. + beta * sin( 2. * th );
    double a    = 1. / zeta;
    double z    = -a * exp( -a );
    double W    = lambertW_1( z );
    return ( sqrt( -W * zeta ) - 1. ) / ( B * v0 * cos( th ) );
}

double v_bar( double G, double B, double v0, double th  ) {   // average speed over entire flight

    double u0   = v0 * cos( th );
    double beta = B * v0 * v0 / G;
    double zeta = 1. + beta * sin( 2. * th );
    double a    = 1. / zeta;
    double z    = -a * exp( -a );
    double W    = lambertW_1( z );
    return u0 * ( sqrt( -W * zeta ) + 1. ) / ( 2. * zeta );
}

```

```
72  double u_impact( double G, double B, double v0, double th ) {   // x-component of velocity at impact (m/s)
73
74      double u0   = v0 * cos( th );
75      double beta = B * v0 * v0 / G;
76      double zeta = 1. + beta * sin( 2. * th );
77      double a    = 1. / zeta;
78      double z    = -a * exp( -a );
79      double W    = lambertW_1( z );
80      return u0 / sqrt( -W * zeta );
81  }
82
83  double w_impact( double G, double B, double v0, double th ) {   // y-component of velocity at impact (m/s)
84
85      double w0   = v0 * sin( th );
86      double beta = B * v0 * v0 / G;
87      double zeta = 1. + beta * sin( 2. * th );
88      double a    = 1. / zeta;
89      double z    = -a * exp( -a );
90      double W    = lambertW_1( z );
91      return w0 * ( 1. + ( W * zeta + 1. ) / ( zeta - 1. ) ) / sqrt( -W * zeta );
92  }
93
94  double v_impact( double G, double B, double v0, double th ) {   // speed at impact (m/s)
95
96      double u = u_impact( G, B, v0, th );
97      double w = w_impact( G, B, v0, th );
98      return sqrt( u * u + w * w );
99  }
100
101 double th_impact( double G, double B, double v0, double th ) {   // angle at impact (rad)
102
103     double beta = B * v0 * v0 / G;
104     double zeta = 1. + beta * sin( 2. * th );
105     double a    = 1. / zeta;
106     double z    = -a * exp( -a );
107     double W    = lambertW_1( z );
108     return atan( ( 1. + ( W * zeta + 1. ) / ( zeta - 1. ) ) * tan( th ) );
109 }
110
111 double u( double B, double v0, double th, double t ) {   // x-component of velocity (m/s)
112
113     double u0 = v0 * cos( th );
114     return u0 / ( 1. + B * u0 * t );
115 }
116
117 double x_time( double B, double v0, double th, double t ) {   // distance downrange (m)
118
119     double u0 = v0 * cos( th );
120     return log( 1. + B * u0 * t ) / B;
121 }
122
123 double w( double G, double B, double v0, double th, double t ) {   // y-component of velocity (m/s)
124
125     double u0 = v0 * cos( th );
126     double w0 = v0 * sin( th );
127     double c  = 0.5 * G * t;
128     return ( w0 - c ) / ( 1. + B * u0 * t ) - c;
129 }
130
131 double y_time( double G, double B, double v0, double th, double t ) {   // height (m)
132
133     double u0 = v0 * cos( th );
134     double w0 = v0 * sin( th );
135     double c  = G  / ( 2. * B * u0 );
136     return ( w0 + c ) * log( 1. + B * u0 * t ) / ( B * u0 ) - c * t - 0.25 * G * t * t;
137 }
138
139 int main( int argc, char* argv[] ) {
140
141     const double D2R     = M_PI / 180.;                              // to convert degrees to radians
142     const double R2D     = 180. / M_PI;                              // to convert radians to degrees
143     const double G       = 9.80665;                                  // acceleration due to gravity (m/s^2)
144     const double RHO_AIR = 1.205;                                    // density of air at 20 deg Celsius (kg/m^3)
145
146     const double m       = 1.;                                       // mass of projectile (kg)
147     const double rho     = 7830.;                                    // density of steel (kg/m^3)
148     const double D       = pow( 6. * m / ( M_PI * rho ), 1./3. );    // diameter of spherical projectile (m)
149     const double Ap      = 0.25 * M_PI * D * D;                      // presented area of spherical projectile (m^2)
150     const double Cd      = 0.5;                                      // drag coefficient (dimensionless)
151     const double B       = RHO_AIR * Cd * Ap / ( 2. * m );           // m^-1
152
153     double v0 = 1000.;                        // launch speed (m/s)
154     double deg = 5.;                          // default launch angle (deg)
155     if ( argc == 2 ) deg = atof( argv[1] );   // launch angle specified on command line (deg)
156     double th = deg * D2R;                    // launch angle converted to rad (rad)
157
158     double beta = B * v0 * v0 / G;            // dimensionless ratio of deceleration due to air drag and acceleration due to gravity
159     double R = range( G, B, v0, th );         // range (m)
160     double T = t_range( G, B, v0, th  );      // total time of flight (s)
161
162     cout << "Launch angle (deg)                    = " << deg                              << endl
163          << "b (m^-1)                              = " << B                                << endl
164          << "beta (-)                              = " << beta                             << endl
165          << "Range of trajectory (m)               = " << R                                << endl
166          << "Maximum height of trajectory (m)      = " << height( G, B, v0, th )           << endl
167          << "Elapsed time to achieve max height (s) = " << t_height( G, B, v0, th )        << endl
168          << "Downrange distance at max height (m)  = " << x_height( G, B, v0, th )         << endl
169          << "Total time of flight (s)              = " << T                                << endl
170          << "Average speed over entire flight (m/s) = " << v_bar( G, B, v0, th )           << endl
171          << "Impact speed (m/s)                    = " << v_impact( G, B, v0, th )         << endl
172          << "Impact angle (deg)                    = " << th_impact( G, B, v0, th ) * R2D << endl;
173
174     ofstream fout( "quadratic.out" );
175     for ( double x = 0.; x < R; x += R / 999. ) fout << x << "   " << y( G, B, v0, th, x ) << endl;
176     fout << R << "\t" << 0. << endl;
177     fout.close();
178
```

```
    return EXIT_SUCCESS;
}
```

Implementation of the Runge-Kutta [20, 3] program used to test the approximate closed-form solution discussed in Section 5.

Listing 5. rk.cpp

```
// rk.C: Runge-Kutta fourth-order method for projectile motion under influence of gravity and air resistance
//       Adapted from Richard Sandmeyer's program (SI Units)
// R. Saucier, July 2012

#include <iostream>
#include <fstream>
#include <cstdlib>
#include <cmath>
using namespace std;

int main( int argc, char* argv[] ) {

    const double D2R     = M_PI / 180.;   // to convert deg to rad
    const double R2D     = 1. / D2R;      // to convert rad to deg
    const double G       = 9.80665;       // acceleration due to gravity at sea level (m/s^2)
    const double RHO_AIR = 1.205;         // air density at 20 deg Celsius (kg/m^3)
    const double h       = 0.0005;        // time step (s)
    const double XOUT    = 1.;            // output approximately every 1 m downrange
    const double YSTOP   = 0.;            // cutoff altitude (m)
    double rho, m, b, beta, c, v, xnew, u, unew, xout, y, ynew, wnew, w, d, cd, ap, x, t, deg, theta;
    double k1x, k2x, k3x, k4x, l1x, l2x, l3x, l4x, k1y, k2y, k3y, k4y, l1y, l2y, l3y, l4y;

    rho   = 7830.;                                  // density of steel (kg/m^3)
    m     = 1.;                                     // mass of projectile (kg)
    d     = pow( 6. * m / ( M_PI * rho ), 1./3. );  // diameter of spherical projectile (m)
    ap    = 0.25 * M_PI * d * d;                    // presented area of spherical projectile (m^2)
    cd    = 0.5;                                    // drag coefficient (dimensionless)
    v     = 1000.;                                  // initial velocity (m/s)
    deg   = 5.;                                     // default launch angle (deg)
    if ( argc == 2 ) deg = atof( argv[1] );         // launch angle specified on command line (deg)
    theta = deg * D2R;                              // convert launch angle from deg to radians

    u = v * cos( theta );   // x component of velocity (m/s)
    w = v * sin( theta );   // y component of velocity (m/s)
    x     = 0.;
    y     = 0.;
    t     = 0.;
    xout  = XOUT;

    // fourth-order Runge-Kutta method inside this "while" loop (fixed step size, not adaptive step size)

    double height = 0., t_height = 0., x_height = 0.;
    b    = RHO_AIR * cd * ap / ( 2. * m );
    beta = b * v * v / G;

    ofstream fout( "rk.out" );
    while ( y > -0.001 ) {

       k1x  = h * u;
       k1y  = h * w;
       v    = sqrt( u * u + w * w );
       c    = -b * v;
       l1x  = h * ( c * u );
       l1y  = h * ( c * w - G );

       k2x  = h * ( u + 0.5 * l1x );
       k2y  = h * ( w + 0.5 * l1y );
       unew = u + 0.5 * l1x;
       wnew = w + 0.5 * l1y;
       v    = sqrt( unew * unew + wnew * wnew );
       c    = -b * v;
       l2x  = h * ( c * unew );
       l2y  = h * ( c * wnew - G );

       k3x  = h * ( u + 0.5 * l2x );
       k3y  = h * ( w + 0.5 * l2y );
       unew = u + 0.5 * l2x;
       wnew = w + 0.5 * l2y;
       v    = sqrt( unew * unew + wnew * wnew );
       c    = -b * v;
       l3x  = h * ( c * unew );
       l3y  = h * ( c * wnew - G );

       k4x  = h * ( u + l3x );
       k4y  = h * ( w + l3y );
       unew = u + l3x;
       wnew = w + l3y;
       v    = sqrt( unew * unew + wnew * wnew );
       c    = -b * v;
       l4x  = h * ( c * unew );
       l4y  = h * ( c * wnew - G );

       xnew = x + ( k1x + k2x + k2x + k3x + k3x + k4x ) / 6.;
       ynew = y + ( k1y + k2y + k2y + k3y + k3y + k4y ) / 6.;
       unew = u + ( l1x + l2x + l2x + l3x + l3x + l4x ) / 6.;
       wnew = w + ( l1y + l2y + l2y + l3y + l3y + l4y ) / 6.;

       t += h;
       x  = xnew;
       y  = ynew;
       u  = unew;
       w  = wnew;
       v  = sqrt( u * u + w * w );

```

```
95          if ( y > height ) {
96             height   = y;
97             t_height = t;
98             x_height = x;
99          }
100
101         if ( y <= YSTOP ) {
102            x -= y * u / w;
103            y  = YSTOP;
104            fout << x << "\t" << y << endl;
105            break;
106         }
107
108         if ( x >= xout ) {
109            fout << x << "\t" << y << endl;
110            xout += XOUT;
111         }
112      }
113      fout.close();
114
115      cout << "Launch angle (deg)                    = " << deg                  << endl
116           << "b (m^-1)                              = " << b                    << endl
117           << "beta (-)                              = " << beta                 << endl
118           << "Range of trajectory (m)               = " << x                    << endl
119           << "Maximum height of trajectory (m)      = " << height               << endl
120           << "Elapsed time to achieve max height (s) = " << t_height            << endl
121           << "Downrange distance at max height (m)  = " << x_height             << endl
122           << "Total time of flight (s)              = " << t                    << endl
123           << "Average speed over entire flight (m/s) = " << x / t               << endl
124           << "Impact speed (m/s)                    = " << sqrt( u * u + w * w ) << endl
125           << "Impact angle (deg)                    = " << atan( w / u ) * R2D   << endl;
126
127      return EXIT_SUCCESS;
128   }
```