AerE 161
Project #2: Flight Path of a Projectile

Hammad Imam
Undergraduate, Aerospace Engineering
Iowa State University

March 23rd, 2018

Contents

1 Problem Statement

The purpose of this project was to study the effects of air resistance on the flight path of projectiles. Since objects travel differently with and without air resistance, two formulas were used to calculate trajectories. This project utilizes 5 different .m files to calculate the data, plot different graphs, and to pass values to be calculated and plotted. For the output, a struct was created with data on the trajectory of the object, and 6 graphs were generated based off the data in the struct.

2 Theory

2.1 Introduction and Symbols Used

In this section, the methods of calculation for the graphs will be discussed. The first step will be explaining the approach taken to calculate the values. After that, the equations will be given and any calculations needed to use them in a way different than the original form will be explained, first for projectiles without air resistance, then for projectiles with air resistance. All calculations used in the code will be explained in this section.
The following symbols will be used in this section

- t - time, [s]
- k - Coefficient of Resistance, [1/s]
- x - Horizontal Position or Distance, [m]
- y - Vertical Position or Altitude, [m]
- v_0 - Launch velocity, [m/s]
  - v_{0x} - x component of launch velocity, [m/s]
  - v_{0y} - y component of launch velocity, [m/s]
- θ - Launch angle from the horizontal, [deg]
- g - Gravitational acceleration constant, 9.81 m/s^2

2.2 Approach

When finding the trajectory, both position and velocity in the x and y directions need to be calculated. All of these can be found as functions of time, so the first step in generating these values will be to find the time that the object spends travelling. After the time it takes to travel the whole distance is found, the time can be stepped up from 0 to its maximum value, and the kinematic equations can be found in terms of time, t. These can all be incorporated into vectors such that the nth element of any vector corresponds to the value of that vector at the time at the nth element of the time vector. All of these vectors will be incorporated into a struct for a single value of the air resistance coefficient, k. These structs will then be combined into a single struct array containing all kinematic values for the given values of k.

2.3 Finding Total Time Elapsed

An object is "in the air" from when it leaves the ground to when it returns, so it follows that the total time elapsed will be at the instant that the object hits the ground. In more algebraic terms, if $y(t)$ is the height of the object as a

function of time, then $y(t_{\max}) = 0$. To find this, we will use the equation for y, set it equal to 0, and solve for t.

$$\begin{aligned} y(t) &= v_0 sin(\theta)t - \frac{1}{2}gt^2 \\ 0 &= t(v_0 sin(\theta) - \frac{g}{2}t) \\ t &= 0, t = \frac{2v_0 sin(\theta)}{g} \end{aligned}$$

As shown above, solving for t gives us two values. The first one $t = 0$ makes sense because it's at the time the projectile is being launched; the second point is the one we're more interested in because it tells us when the projectile lands. This gives us the equation for time elapsed from launch to landing, so our equation for time elapsed is

$$t = \frac{2v_0 sin(\theta)}{g} \tag{1}$$

The issue now comes up that this t value is calculated using the y equation without air resistance being taken into account. Since the air resistance prevents the object from gaining as much speed as it would without air resistance, we can safely make the assumption that the time elapsed with air resistance will be the same or lower than the time elapsed without. In the code, since we are stepping the t value up and calculating values based on that, we can cover all cases by stopping to step t up either at the calculated value for t_{max} or as soon as the calculated value for y is equal to 0.

Since the t portion of the calculation is taken care of, the equations for the kinematic components can be listed and we can begin to use them in the calculations.

2.4 Without Air Resistance ($k = 0$)

2.4.1 Horizontal Velocity, v_x, m/s

$$v_x(t) = v_0 cos(\theta) \tag{2}$$

2.4.2 Vertical Velocity, v_y, m/s

$$v_y(t) = v_0 sin(\theta) - gt \tag{3}$$

2.4.3 Distance, x, m

$$x(t) = v_0 cos(\theta)t \tag{4}$$

2.4.4 Altitude, y, m

$$y(t) = v_0 sin(\theta)t - \frac{1}{2}gt^2 \tag{5}$$

2.5 With Air Resistance ($k \neq 0$)

2.5.1 Horizontal Velocity, v_x, m/s

$$v_x(t) = v_0 cos(\theta)e^{-kt} \tag{6}$$

2.5.2 Vertical Velocity, v_y, m/s

$$v_y(t) = v_0 sin(\theta)e^{-kt} + \frac{g}{k}(e^{-kt} - 1) \tag{7}$$

2.5.3 Distance, x, m

$$x(t) = \frac{v_0 cos(\theta)}{k}(1 - e^{-kt} \tag{8}$$

2.5.4 Altitude, y, m

$$y(t) = -\frac{gt}{k} + \frac{kv_0 sin(\theta) + g}{k^2}(1 - e^{-kt}) \tag{9}$$

2.6 Conclusion

Now that we have all our equations, it's time to make sure that we have all our variables in order to calculate what we need. g is a known constant, v_0 and θ will be given in the problem, and t and k will be produced by the code. Since we have everything we need, it's safe to move forwards and use the code to generate results.

3 Solution

3.1 Overview

This section contains the solution to the problem, which is 5 .m files and 6 graphs. The .m files all contain comments at the start that describe the purpose of that file, and then more comments in the body that describe what the code is doing. Additionally, short description is provided before each screenshot to describe what is being shown.

3.2 Code

3.2.1 flightpath.m

This is the main function that provides the data used for other .m files. It receives values for v_0, θ, k. The data is provided in a 1×n struct where n is the length of the k value that is passed to it. Each element of the struct contains vectors for t, v_x, v_y, x, and y.

```
%Hammad Imam // himam@iastate.edu
%AERE 161 Project 2
%Function
%Calculates the characteristics of the projectile flightpath.
%Input: v0, theta, k
%Output: struct containing t, x, y, u, v vectors for each k

function fs = flightpath(v0, theta, k)

    %assign constants to be used
    g = 9.81; %gravitational constant
    res = 1000; %resolution of data
    %split v into x and y components
    v_x = v0*cosd(theta);
    v_y = v0*sind(theta);
    tmax = (2*v0*sind(theta))/g; %find the max time

    for i = 1:size(k,2) %for every item in k, create a new struct element
        fs(i).k = k(i); %assign the struct element the value of k

        if k(i) == 0    %for the case with no air resistance
            for j = 1:res
                %step the calculation to create res number of data pts
                t = (j/res)*tmax;
                %increment time in small steps proportional to max time
                fs(i).t(j) = t;
                %set each element of the time vector to stepped time

                %velocity formulas
                fs(i).u(j) = v_x;
                fs(i).v(j) = v_y - g*t;

                %position formulas
                fs(i).x(j) = v_x * t;
                fs(i).y(j) = v_y*t - .5*g*(t^2);
            end
        else %if there is some air resistance
            for j = 1:res
                t = j*tmax/res;
                fs(i).t(j) = t;
                %similar setup to the ones above

                y = -g*t/k(i) + (k(i)*v_y + g)/( (k(i))^2 ) * (1 - exp(-k(i)*t));
                %find the altitude given time

                if y < 0    %in the case that the object lands on the ground
                    fs(i).t= fs(i).t(1:end-1); %remove the last data pt for time
                    break %end the for loop
                end

                %velocity formulas
                fs(i).u(j) = v_x * exp(-k(i)*t);
                fs(i).v(j) = v_y * exp(-k(i)*t) + (g/k(i))*(exp(-k(i)*t) -1);

                %position formulas (y was solved for earlier)
                fs(i).y(j) = y;
                fs(i).x(j) = (v_x / k(i))*(1 - exp(-k(i)*t));
            end
        end
    end
end
```

3.2.2 plot_flightpaths.m

This is a function that takes the struct generated by flightpath.m and outputs 4 graphs: $y(x)$, $y(t)$, $v_x(t)$, and $v_y(t)$.

```
%Hammad Imam // himam@iastate.edu
%AERE 161 Project 2
%Function
%Plots y(x), y(t), u(t), and v(t)

function plot_flightpaths(fs)
%create empty strings that will be filled with code to be run
p = '';
l = '';

%Altitude, y / Distance, x
figure
for i = 1:size(fs,2) %for every part of the struct
    p = strcat(p,sprintf('fs(%d).x, fs(%d).y,',i,i));
    %extract the x and y vectors and append them to p
    l = strcat(l, sprintf('num2str(fs(%d).k),',i));
    %extract the k value to fill in the legend
end
pl = strcat('plot(',p(1:end-1),');');
%use p to generate the items in the plot, strcat to pl to make it run
leg = strcat('lgd = legend(',l(1:end-1),');');
%use l to generate a legend
eval(pl); %generate the plot with all the different x/y pairs
xlabel('Distance, x, [m]');  %label the x
ylabel('Altitude, y, [m] '); %label the y
title('Altitide vs Distance'); %give it a title
eval(leg); %generate the legend
title(lgd, 'k, [1/s]'); %title the legend

%everything after this is similar to the first example

%Altitude, y / Time, t
figure
p = '';
for i = 1:size(fs,2)
    p = strcat(p,sprintf('fs(%d).t, fs(%d).y,',i,i));
end
```

```
pl = strcat('plot(',p(1:end-1),');');
eval(pl);
xlabel('Time, t, [s]');
ylabel('Altitude, y, [m] ');
title('Altitide vs Time');
eval(leg);
title(lgd, 'k, [1/s]');

%Horizontal Velocity, v_x / Time, t
figure
p = '';
for i = 1:size(fs,2)
    p = strcat(p,sprintf('fs(%d).t, fs(%d).u,',i,i));
end
pl = strcat('plot(',p(1:end-1),');');
eval(pl);
xlabel('Time, t, [s]');
ylabel('Horizontal Velocity, v_x, [m/s] ');
title('Horizontal Velocity vs Time');
eval(leg);
title(lgd, 'k, [1/s]');

%Vertical Velocity, u / Time, t
figure
p = '';
for i = 1:size(fs,2)
    p = strcat(p,sprintf('fs(%d).t, fs(%d).v,',i,i));
end
pl = strcat('plot(',p(1:end-1),');');
eval(pl);
xlabel('Time, t, [s]');
ylabel('Vertical Velocity, v_y, [m/s] ');
title('Vertical Velocity vs Time');
eval(leg);
title(lgd, 'k, [1/s]');
```

3.2.3 main_flightpaths.m

This is a script that gives initial values to flightpath.m, takes that data, and passes it to plot_flightpaths.m to output graphs of the flightpath, altitude over time, and horizontal and vertical velocities.

```
%Hammad Imam // himam@iastate.edu
%AERE 161 Project 2
%Script
%Initializes parameters and calls flightpath.m and plot_flightpaths.m

clear, clc;

%Initializing parameters
v0 = 600;
theta = 60;
k = [0 0.005 0.01 0.02 0.04 0.08];

%call flightpath.m, feed data to plot_flightpaths.m
fs = flightpath(v0,theta,k);
plot_flightpaths(fs);
```

3.2.4 plot_range.m

This is a function that takes the struct generated by flightpath.m and outputs 2 graphs: $x(k)$, $t_{max}(k)$.

```
%Hammad Imam // himam@iastate.edu
%AERE 161 Project 2
%Function
%Plots x(k) and t_max(k)

function plot_range(fs)

k = [];
x = [];
t = [];

%gather k, tmax, and xmax into vectors
for i = 1:size(fs,2)
    k(i) = fs(i).k;
    x(i) = fs(i).x(end);
    t(i) = fs(i).t(end);
end

%Range x, Resistance k
figure
plot(k, x);
xlabel('Coefficient of Resistance, k, [1/s]');
ylabel('Range, x_m_a_x, [m]');
title('Range vs Coefficient of Resistance');

%Total Time t, Resistance k
figure
plot(k, t);
xlabel('Coefficient of Resistance, k, [1/s]');
ylabel('Total Time, t_m_a_x, [s]');
title('Total Time vs Coefficient of Resistance');
```

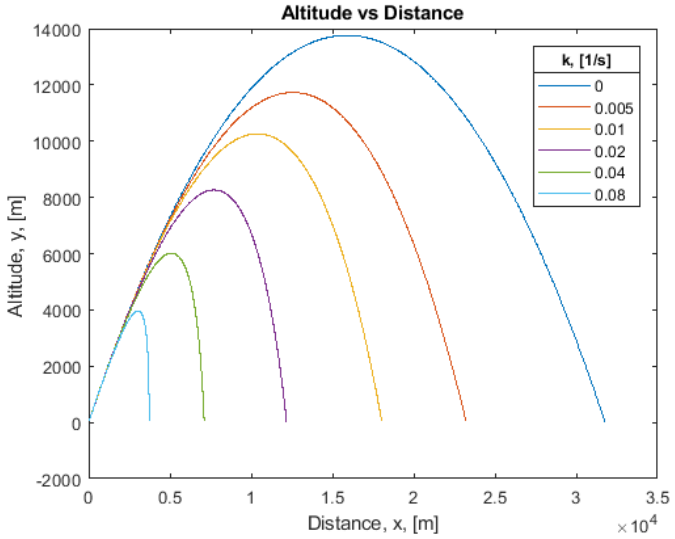
3.2.5 main_range.m

This is a script that gives initial values to flightpath.m, takes that data, and passes it to plot_range.m to output graphs of the range and time as functions of k.

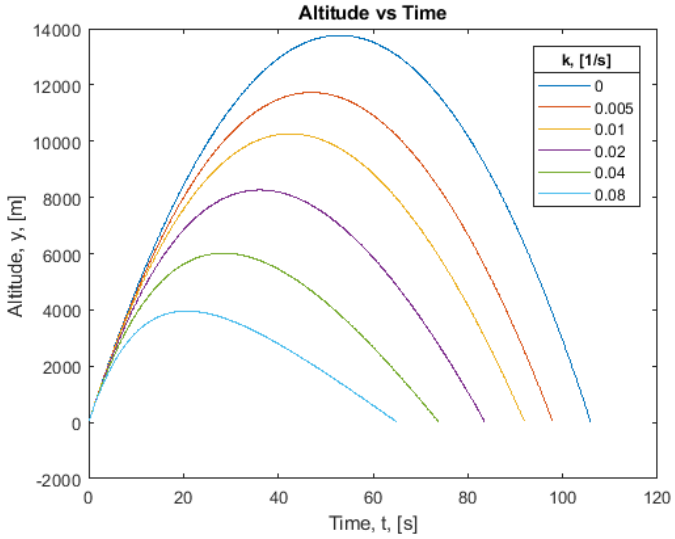
```
%Hammad Imam // himam@iastate.edu
%AERE 161 Project 2
%Script
%Initializes parameters and calls flightpath.m and plot_range.m

clear, clc;

%constant values used throughout
v0 = 600;
theta = 60;
k = (0:0.001:0.08);

%call flightpath.m, use value to plot range.
fs = flightpath(v0,theta,k);
plot_range(fs);
```

3.3 Plots

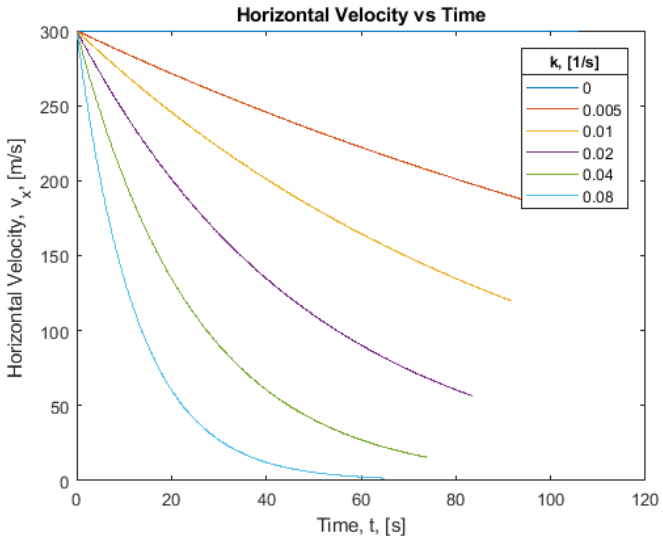
3.3.1 Altitude y vs. Distance x



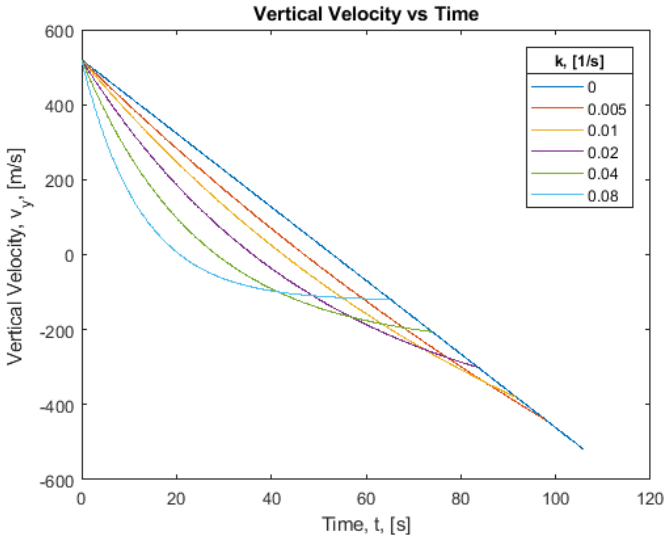
3.3.2 Altitude y vs. Time t



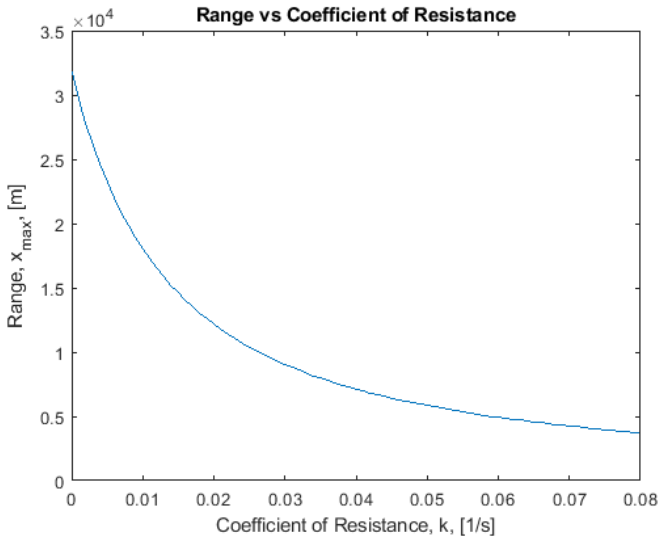
3.3.3 Horizontal Velocity v_x vs. Time t



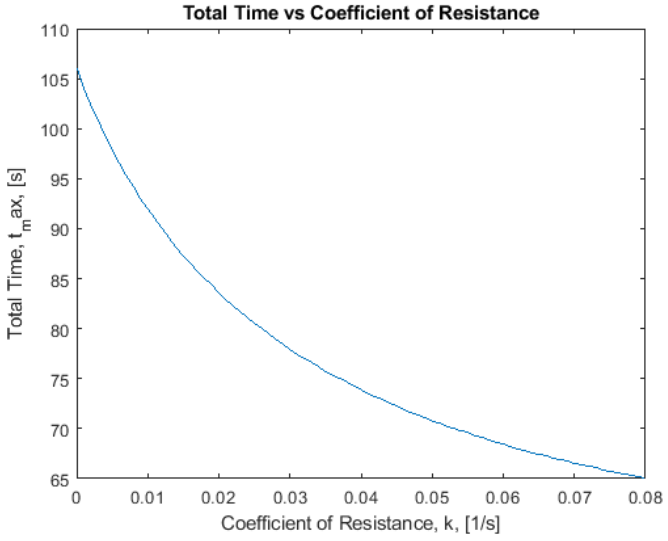
3.3.4 Vertical Velocity v_y vs. Time t



3.3.5 Range x_{max} vs. Coefficient of Resistance k



3.3.6 Time t_{max} vs. Coefficient of Resistance k



4 Discussion

4.1 Analysis

From the graphs produced, we can draw some conclusions about how the projectile behaves. As the coefficient of air resistance increases, both time spent in the air and the range of the projectile decrease, with the range showing a greater decrease. This is because some of the object's kinetic energy is wasted pushing through the air and therefore lost to drag. As a result, the highest altitude reached is also lower. The effects on velocity are interesting, as increasing the coefficient of air resistance causes the vertical velocity to drop faster at the start, but then end along the same line as when there was no air resistance. The horizontal velocity showed that higher k values would lose speed faster, but then hold onto the speed at a more gradual rate, as opposed to the horizontal velocity without any air resistance, which was constant.

4.2 Challenges

My initial approach to this problem was to find the final time and step to that for all values. However, algebraically manipulating the expressions with air resistance proved difficult, so the code was changed to continue advancing until the altitude reached 0. My method of doing this involved using points evenly spaced on some interval, so there was another challenge with finding a number of points that smoothly showed everything, as the resolution required to accurately depict something on some graphs was too low to depict things on other graphs. I ended up playing with the resolution settings and ultimately kept it set pretty high despite the fact that this made generating results take a second or two longer since the calculations became more memory intensive. The biggest problem I had was how to display the data since it needed to be returned as a vector of vectors for every given k value, so I ended up choosing a struct because it was easy for me to mentally associate the k value with the set of vectors. This again caused problems as it quickly became more memory intensive than I had intended, but it worked out fine.

This project is available on GitHub:
`https://github.com/himam99/AERE161-Project1`