Realistic projectile motion

Lorenzo Pantolini

September 17, 2018

Abstract

In this work we consider the motion of a particle subject to drag force. Then we numerically compute the trajectory of the particle for a given initial velocity and we compare the results with the same problem but in vacuum. In the end we estimate the direction of the initial velocity needed to hit a specific target.

1 Introduction

Consider the motion of a projectile thrown into the air, subject to only the acceleration of gravity. Assuming constant gravitational acceleration and negligible air resistance, projectile motion is analytically solvable. Its solutions are expressible in closed-form:

$$\begin{cases} x = x_0 + v_{0x}t \\ y = y_0 + v_{0y}t - \frac{1}{2}gt^2 \end{cases} \tag{1}$$

Where (x_0, y_0) and $\vec{v_0} = (v_{0x}, v_{0y})$ are the initial conditions, t is the time and g is the acceleration of gravity. However, to describe the problem more realistically, we need to consider the effects of air resistance, or drag. The air resistance generates a force acting opposite to the velocity of the object. We can approximate the magnitude of this resistance as:

$$F_d = Av + Bv^2 \tag{2}$$

Where A and B are constants depending on both the body and the fluid characteristics. There exists a critic value of the velocity v_c such that: for $v < v_c$ the term proportional to the velocity prevails while for $v > v_c$ the second term prevails. It is experimentally found that v_c depends on the density and the viscosity of the fluid, and from the dimensions and the shape of the body. We know that for most of the bodies moving in the air, the critical velocity is very low so that the drag force can be considered proportional to v^2, in vector form:

$$\vec{F}_d = -Bv\vec{v} \tag{3}$$

The inclusion of these effects renders the problem analytically unsolvable, and no closed-form solutions are known except under limited conditions. We can express our problem as:

$$\begin{cases} \ddot{x} = F_{d,x} \\ \ddot{y} = F_{d,y} - g \end{cases} \tag{4}$$

In this work we will use $B = 4 \cdot 10^{-5}\ m^{-1}$ and $g = 9,81\ m/s^2$.

2 Compute the trajectory

Since the problem is analytically unsolvable we need a numerical method to compute the trajectory. We can rewrite our system as a coupled set of first-order ODEs:

$$\begin{cases} \dot{x} = v_x \\ \dot{v}_x = -Bvv_x \\ \dot{y} = v_y \\ \dot{v}_y = -Bvv_y - g \end{cases} \tag{5}$$

or in a more compact form:

$$\frac{d\vec{Y}}{dt} = \vec{R} \tag{6}$$

Then, given the initial condition $\vec{Y}_0 = \vec{Y}(t = t_0)$ we consider an interval $[0, t_e]$ which represents our domain of integration. In our case t_e is the time at which the projectile reaches the ground. The general strategy is to divide the interval into steps of equal size h and then develop a recursive formula relating $\vec{Y}(t_n)$ with $\vec{Y}(t_{n-1})$, where $t_n = nh$. To achieve a high level of accuracy we can formally integrate the ODE and rewrite it as:

$$Y_n = Y_{n-1} + \int_{t_{n-1}}^{t_n} R(t, Y)dt \tag{7}$$

As recursive formula we used a *Runge-Kutta* method. Runge-Kutta methods are single-step algorithms as they require information only in the current time interval and not from previous ones. They are based on Taylor expansion and yield better algorithms in terms of accuracy and stability. We used the fourth-order Runge-Kutta which approximates the integral using the *Simpson's Rule*. It requires 4 function evaluations per step and has

local accuracy of $O(h^5)$. For each step we compute the quantities:

$$\begin{cases} \vec{k}_1 = \vec{R}(t_n, \vec{Y}_n) \\ \vec{k}_2 = \vec{R}(t_n + \frac{h}{2}, \vec{Y}_n + \frac{h}{2}\vec{k}_1) \\ \vec{k}_3 = \vec{R}(t_n + \frac{h}{2}, \vec{Y}_n + \frac{h}{2}\vec{k}_2) \\ \vec{k}_4 = \vec{R}(t_n + h, \vec{Y}_n + h\vec{k}_3) \end{cases} \tag{8}$$

then the recursive formula is:

$$\vec{Y}_{n+1} = \vec{Y}_n + \frac{h}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) \tag{9}$$

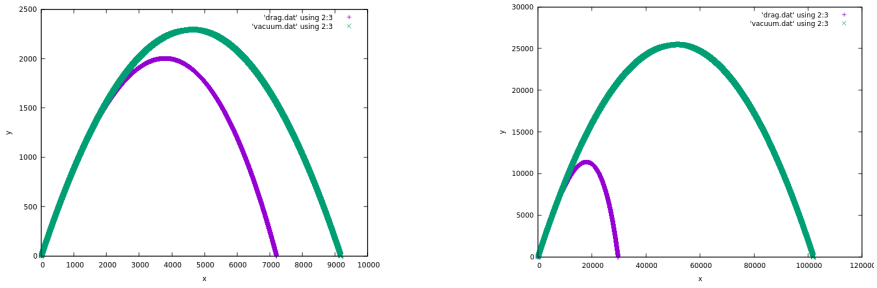

Figure 1: Here $\vec{v}_0$ forms an angle $\theta = \frac{\pi}{4}$ with the x axis for both the plots. The magnitude instead is $v_0 = 300\ m/s$ for the plot on the left and $v_0 = 1000\ m/s$ for the right one.

On the plots we show the different trajectories of the projectile in vacuum and in air for two different initial velocities. As expected the range of the projectile in air is smaller than the one in vacuum considering the same $\vec{v_0}$. Furthermore we notice that the effect of the drag force is stronger for bigger velocities. In order to observe how the drag effect grows with the magnitude of the initial velocity we can show the ratio between the distance traveled in air and in vacuum: $D = \frac{d_{air}}{d_{vacu}}$ over v_0:

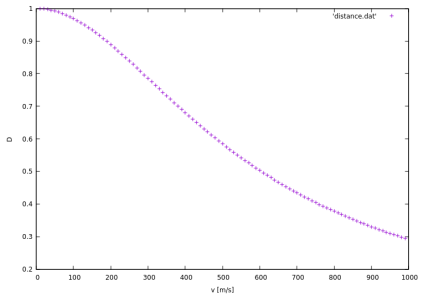

Figure 2: All the trajectories were computed with the same initial angle $\theta = \frac{\pi}{4}$.

We observe that for low values of $v_0 \to D \approx 1$ so there is not a big difference between the trajectories, while for high velocities the range in vacuum is significantly bigger.

3 Maximum range

We know that in vacuum in order to reach the maximum range we have to orient our "cannon" at $\theta_{max} = \frac{\pi}{4}$. Since the drag effect changes the trajectory, which is no more parabolic, also θ_{max} will change.

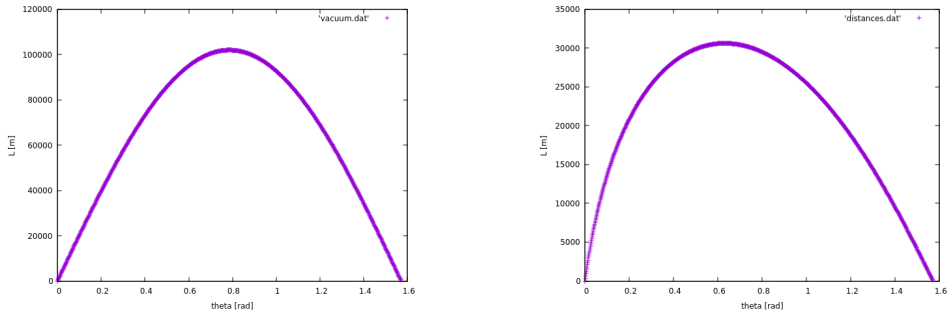

Figure 3: Here we show the distance traveled in function of θ. The plot on the left is in vacuum while the one on the right is in air. In both the cases $v_0 = 1000\ m/s$.

We can also study the case in which there is a difference in altitude between the cannon and the target. In this case we will have $\theta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

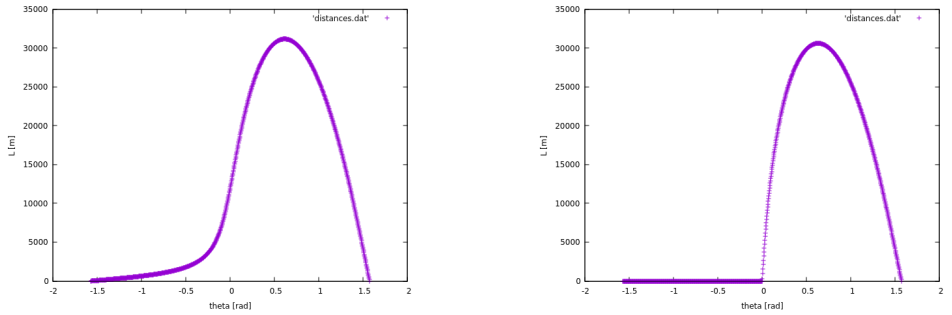

Figure 4: The plots show again the distance traveled in function of θ (both in air), on the right the cannon is on the ground, while on the left it is 1000 m over the target.

Notice that for negative angles the distance traveled is considerably lower than for the positive ones. Plotting θ_{max} over v_0 we observe that the angle for the maximum range decreases with the magnitude of the initial velocity:

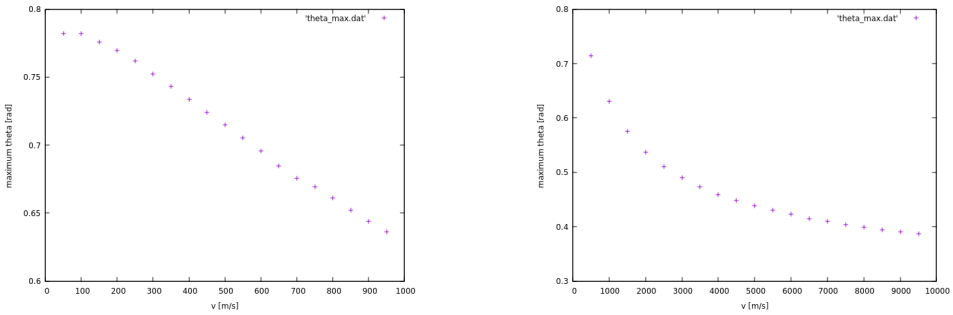

Figure 5: The plots show θ_{max} in two different velocity scales.

4 Shooting method

Now, given the magnitude of the initial velocity v_0 and the distance L to a target, we want to determine the angles (if any) you must orient your cannon at in order to hit the objective. This is a boundary value problem, and given $X(t) = (x(t), y(t))$ our boundary conditions are:

$$X(0) = (0, 0) \qquad X(t_{end}) = (L, 0) \tag{10}$$

It can be useful to express the problem in terms of θ, the angle between the initial velocity and the x axis:

$$\begin{cases} \ddot{x} = -Bv^2 cos(\theta) \\ \ddot{y} = -Bv^2 sin(\theta) - g \end{cases} \tag{11}$$

In order to find the right θ we use the *shooting method*. It consists on turning our problem into a root search for the function:

$$F(\theta) = x(t_{end}, \theta) - L \qquad \text{with} \qquad \theta \in [0, \frac{\pi}{2}] \tag{12}$$

since we want $x(t_{end}, \theta) = L$. We can compute $x(t_{end}, \theta)$ generating a solution for a given θ by integrating the trajectory as seen in section two. To start the root search we need to pinpoint the intervals where a sign change is detected: we will say that a root is bracketed in the interval $[x_n, x_{n+1}]$ if:

$$f(x_n) \cdot f(x_{n+1}) < 0 \tag{13}$$

we observe that this strategy won't be able to detect the root corresponding to the maximum range L_{max} since in that case $F(\theta)$ does not present any change of sign. Once we find the intervals we use the *bisection method* in order to find the roots. Given an interval $[x_n, x_{n+1}]$ bracketing the root, we estimate the zero as the midpoint of the interval: $x_m = \frac{x_n + x_{n+1}}{2}$, then

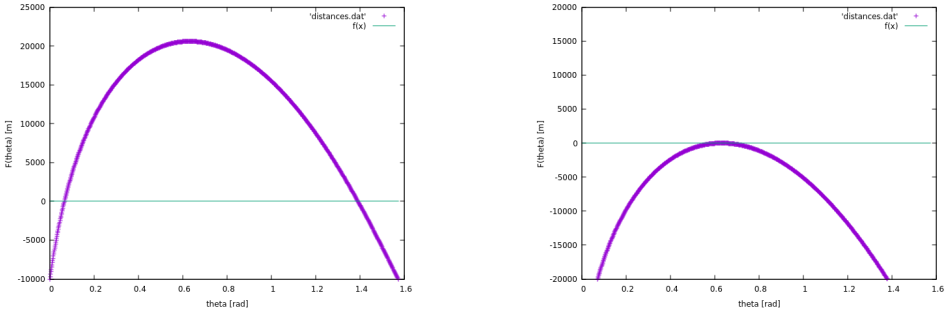

Figure 6: The plots show $F(\theta)$ with $L = 10^4\ m$ on the left and $L = L_{max}$ on the right.

compute $F(x_m)$ and use the midpoint to replace whichever limit has the same sign:

$$\begin{cases} if \quad F(x_n)F(x_m) < 0 \rightarrow x_{n+1} = x_m \\ if \quad F(x_n)F(x_m) > 0 \rightarrow x_n = x_m \end{cases} \tag{14}$$

we repeat this step until convergence is reached. If no intervals bracketing a root are found we need to check if $L = L_{max}$ (maximum range) or if $L > L_{max}$ (out of range). We estimate L_{max} by computing $x(t_{end}, \theta)$ for different θs. Since we know that in air $\theta_{max} < \frac{\pi}{4}$ we start our search from $\frac{\pi}{4}$ and we gradually reduce the angle of $\theta_{inc} = \frac{\pi}{4 \cdot 10^3}$. We stop once found $x(t_{end}, \theta_n)$ such that:

$$x(t_{end}, \theta_n) > x(t_{end}, \theta_{n-1}) \quad and \quad x(t_{end}, \theta_n) > x(t_{end}, \theta_{n+1}) \tag{15}$$

Then we compare L_{max} with the distance of the target to check if their difference is less than a tolerance.

5 Conclusions

Concluding we were able to calculate the trajectory of a projectile in the air and determine the angle of inclination needed to hit a target, all without solving the problem analytically. We also pointed up some differences between the trajectories in air and in vacuum, with particular consideration to the maximum range, which slowly decreases with the increase of the initial speed. We would like to note that the algorithms used to solve our problems can be exchanged depending on the necessities. For example we can trade efficiency with accuracy replacing the fourth-order *Runge-Kutta* method, which is accurate but also computationally expensive, with a faster algorithm like the *Runge-Kutta midpoint* or the *Modified Euler*.

A Code

Here we present the implemented code. Our program is split in three stages, you can choose the one you want to run changing the variable "STAGE" in the source code. The first stage solves the problem, the others were used to produce the plots in this work:

- STAGE 1: given the magnitude of the initial velocity and the distance of a target it returns the angles needed to hit that target. Then it writes the trajectories obtained with those angles in the file "trajectory.dat".
- STAGE 2: for a given v_0 it computes the distances traveled for theta in the interval $[0, \frac{\pi}{2}]$ with a step $\theta_{inc} = \frac{\pi}{10^3}$, then put the results in the file "distances.dat".
- STAGE 3: given v_{max} and v_{inc} it computes the maximum range and the corresponding angle for the velocities between $[0, v_{max}]$ with a step of v_{inc}. Then it writes the results in the file "theta-max.dat".

The functions: "Rkutta4", "Bracket" and "Bisection" are imported by the libraries: "ode-solvers.cxx" and "root-solvers.cxx" through the file "my-header.h".

```
#include "my_header.h"

#define STAGE 1

double B = 4e-05;
double g = 9.81;
double g_v;
double g_L;
double g_altitude = 0;


void Ydot( double t, double *V, double *R );
double Residual( double theta );

int main()
{
        cout << setiosflags(ios::scientific);
        cout << setprecision(12);

#if STAGE == 1
{
        cout <<"Insert the distance from the target:" << endl;
        cin >> g_L;

        cout <<"Insert the magnitude of the initial velocity: " << endl;
        cin >> g_v;

        //variables for the function Bracket
        double a;
        if ( g_altitude > 0 ) a = -M_PI/2;
        else a = 0;

        double b = M_PI/2;
```

```
int N = 100;
int Nr;

double xL[100];
double xR[100];

double xtoll = 1e-10;
double ftoll = 1e-10;

double zero;
int ntry;

//the number of angles is at plus 2
double angles[2];

Bracket(*Residual, a, b, N, Nr, xL, xR);

int find;

for (int i = 0; i < Nr; i++)
{

        find = Bisection(*Residual, xL[i], xR[i], xtoll, ftoll,
            zero, ntry);

        if ( find )
        {
                angles[i] = zero;
                cout <<"In order to hit the target you have to
                    orient the cannon at: " << zero << endl;

        }

        else
        {
                cout <<"No zeros was found" << endl;
        }
}


//check if the target is at maximum range
if( Nr == 0 )
{
        double Nh = 1000;
        //maximum theta in vacuum
        double theta = M_PI/4.0;
        double h = theta/(Nh-1);

        double L = 0;
        double L_max = 0;
        double theta_max;

        for (int i = 0; i < Nh; i++)
        {
                L = Residual(theta) + g_L;

                if ( L > L_max )
                {
                        L_max = L;
                        theta_max = theta;
                }
```

```
                //if decreases I've already found L_max
                else if (L < L_max)
                {
                        break;
                }

                theta -= h;
        }

        double toll_L = L_max*1e-03;
        double diff = fabs(g_L - L_max);

        if ( diff < toll_L )
        {
                angles[0] = theta_max;
                angles[1] = 0;
                cout <<"The target is at the maximum range." <<
                    endl;
                cout <<"You have to orient the cannon at: " <<
                    theta_max << endl;

        }

        else
        {
                angles[0] = 0;
                angles[1] = 0;
                cout <<"The target is out of range"<< endl;
                cout <<"L max = " << L_max << endl;
        }

}

//now we compute the trajectory
ofstream file("trajectory.dat", ios::out);

for (int i = 0; i < 2; i++)
{
        if( angles[i] != 0 )
        {
                int n_eq = 4;
                double Y[n_eq];

                Y[0] = 0;
                Y[1] = g_altitude;
                Y[2] = g_v*cos(angles[i]);
                Y[3] = g_v*sin(angles[i]);

                double t = 0;
                //h = temporal step
                double h = 1e-03;

                do
                {
                        t += h;
                        Rkutta4( *Ydot, t, h, Y, n_eq );

                        file << t <<" "<< Y[0] <<" " << Y[1] <<
                            endl;

                }while( Y[1] > 0 );
        }
```

```
                file <<"\n\n";
        }


}

#endif


#if STAGE == 2
{       //compute distances in function of theta

        cout <<"Insert the magnitude of the initial velocity: " << endl;
        cin >> g_v;

        double Nh = 1000;
        double b = M_PI;
        double h = b/(Nh-1);
        double theta = -M_PI*0.5;
        double L = 0;

        ofstream file("distances.dat", ios::out);

        for (int i = 0; i < Nh; i++)
        {
                L = Residual(theta) + g_L;
                file << theta <<" "<< L << endl;

                theta += h;
        }

}
#endif


#if STAGE == 3
{
        double v_max;
        cout <<"Insert the magnitude of the maximum velocity: " << endl;
        cin >> v_max;

        double v_inc;
        cout <<"Insert the magnitude of the velocity increment: " << endl;
        cin >> v_inc;

        double v = 0;
        double Nh = 1000;
        double b = M_PI/2;
        double h = b/(Nh-1);
        double L;
        double L_max;
        double theta_max;
        double theta = 0;


        ofstream file2("theta_max.dat", ios::out);

        while( v < v_max )
        {
                L_max = 0;
                theta = 0;
```

```
                g_v = v;

                for (int i = 0; i < Nh; i++)
                {
                        L = Residual(theta) + g_L;

                        if (L > L_max)
                        {
                                L_max = L;
                                theta_max = theta;
                        }

                        theta += h;
                }

                file2 << v <<" "<< theta_max <<" "<< L_max << endl;

                v += v_inc;
                cout << v << endl;
        }


}
#endif


return 0;

}


void Ydot( double t, double *Y, double *R )
{
        double vx = Y[2];
        double vy = Y[3];
        double v = sqrt(vx*vx + vy*vy);

        R[0] = vx;
        R[1] = vy;
        R[2] = - B*v*vx;
        R[3] = - g - B*v*vy;


        return;
}


double Residual( double theta )
{
        int n_eq = 4;

        double Y[n_eq];

        Y[0] = 0;
        Y[1] = g_altitude;
        Y[2] = g_v*cos(theta);
        Y[3] = g_v*sin(theta);

        double t = 0;
        double h = 1e-03;

        do
```

```
        {
                t += h;
                Rkutta4( *Ydot, t, h, Y, n_eq );

        }while( Y[1] > 0 );


        return (Y[0] - g_L);

}
```