APPM 3050
Project #1 - Projectile Motion

Fiona Pigott, Dustin Martin, Christopher Miller

April 6, 2012

Abstract

Predicting the trajectory of a projectile through air presents computational and mathematical challenges in that it is impossible to analytically solve for the projectile's position at any given time. The only aspect of the motion of the projectile that is known is the force acting on it, and from there, the acceleration. In order to predict the path of an artillery shot, the acceleration is integrated numerically to give the approximate velocity and position at any given time.

The goal of this project is to analyze the path of a projectile beginning at the origin with enough accuracy to aim it towards a target in the first quadrant of a coordinate axis. This is accomplished by numerically integrating the coupled system of differential equations that define the acceleration and finding where the resultant trajectories pass closest to the target. By using a bisection algorithm to find where the trajectory passes through the target, it is possible to zero in on a firing angle that will hit the target, while accounting for constant wind resistance.

1 Background

Newton's second law governs the motion of moving bodies according to the equation :

$$\sum \mathbf{F} = m\dot{\mathbf{v}}$$

where:

$$\mathbf{v} = \dot{\mathbf{r}}$$

1.1 Physics of a projectile in still air

This equation shows that any change in net force acting on an object will result in a change in its acceleration based on its mass. When a projectile is fired through the air if only has two forces acting on it: the drag force and the force of gravity. The gravitational force will always act towards the earth (which we will denote as a $-\hat{\mathbf{j}}$ direction in our two-dimensional model) and the drag force will always acting in the opposite direction the ball in traveling ($-\hat{\mathbf{T}}$).The resulting sum of forces is:

$$\sum \mathbf{F} = -mg\,\hat{\mathbf{j}} - C_D v^2 \hat{\mathbf{T}} \tag{1}$$

Since the initial velocity the projectile fires at is a constant, we can rewrite the velocity vector based on its dependence on theta in the $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ directions:

$$v = \dot{x} + \dot{y}$$

where:

$$\dot{x} = v\cos(\theta)\,\hat{\mathbf{i}} \tag{2}$$

and:

$$\dot{y} = v\sin(\theta)\,\hat{\mathbf{j}} \tag{3}$$

Finding the derivative of this function with respect to time results in:

$$\dot{v} = [\dot{v}\cos(\theta) - v\sin(\theta)\dot{\theta}]\,\hat{\mathbf{i}} + [\dot{v}\sin(\theta) + v\cos(\theta)\dot{\theta}]\,\hat{\mathbf{j}} \tag{4}$$

Combining equations (1) and (4), we can write a version of Newton's second law specifically applicable to our parameters:

$$m[\dot{v}\cos(\theta) - v\sin(\theta)\dot{\theta}]\,\hat{\mathbf{i}} + m[\dot{v}\sin(\theta) + v\cos(\theta)\dot{\theta}]\,\hat{\mathbf{j}} = -mg\,\hat{\mathbf{j}} - C_D v^2\hat{\mathbf{T}} \tag{5}$$

Currently, this equation is written in terms of the $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$, and $\hat{\mathbf{T}}$ unit vectors. The $\hat{\mathbf{i}}$ and $\hat{\mathbf{j}}$ unit vectors can be written in terms of the tangential ($\hat{\mathbf{T}}$) and normal ($\hat{\mathbf{N}}$) unit vectors based on the θ the projectile was launched at. These relationships are as follows:

$$\hat{\mathbf{i}} = \hat{\mathbf{T}}\cos(\theta) - \hat{\mathbf{N}}\sin(\theta)\,\hat{\mathbf{j}} = \hat{\mathbf{T}}\sin(\theta) + \hat{\mathbf{N}}\cos(\theta) \tag{6}$$

Substituting into equation (5):

$$m[\dot{v}\cos(\theta) - v\sin(\theta)\dot{\theta}](\hat{\mathbf{T}}\cos(\theta) - \hat{\mathbf{N}}\sin(\theta)) + m[\dot{v}\sin(\theta) + v\cos(\theta)\dot{\theta}](\hat{\mathbf{T}}\sin(\theta) + \hat{\mathbf{N}}\cos(\theta)) =$$
$$-mg(\hat{\mathbf{T}}\sin(\theta) + \hat{\mathbf{N}}\cos(\theta)) - C_D v^2\hat{\mathbf{T}} \tag{7}$$

We can set the $\hat{\mathbf{T}}$ and $\hat{\mathbf{N}}$ components on either side of the equation equal to each other in order to determine the unknown variables in terms of the given constants. Let's start by grouping the $\hat{\mathbf{T}}$ components:

$$m\cos(\theta)[\dot{v}\cos(\theta) - v\sin(\theta)\dot{\theta}] + m\sin(\theta)[\dot{v}\sin(\theta) + v\cos(\theta)\dot{\theta}] = -mg\sin(\theta) - C_D v^2$$

$$m\dot{v}\cos^2(\theta) - mv\cos(\theta)\sin(\theta) + m\dot{v}\sin^2(\theta) + mv\cos(\theta)\sin(\theta) = -mg\sin(\theta) - C_D v^2$$

$$m\dot{v}(\cos^2(\theta) + \sin^2(\theta)) = -mg\sin(\theta) - C_D v^2$$

$$\dot{v} = -g\sin(\theta) - \frac{C_D v^2}{m} \tag{8}$$

Now set the $\hat{\mathbf{N}}$ components equal:

$$-m\sin(\theta)[\dot{v}\cos(\theta) - v\sin(\theta)\dot{\theta}] + m\cos(\theta)[\dot{v}\sin(\theta) + v\cos(\theta)\dot{\theta}] = -mg\cos(\theta)$$
$$-\dot{v}\sin(\theta)\cos(\theta) + v\sin^2(\theta)\dot{\theta} + \dot{v}\sin(\theta)\cos(\theta) + v\cos^2(\theta)\dot{\theta} = -g\cos(\theta)$$
$$\dot{\theta}v(\cos^2(\theta) + \sin^2(\theta)) = -g\cos(\theta)$$

$$\dot{\theta} = -\frac{g}{v}\cos(\theta) \tag{9}$$

Equations (8) and (9), combined with the x and y components of the velocity, equations (2) and (3), give us four first order ODEs which can be used to determine the motion of the projectile (without wind).

1.2 Physics of a projectile with wind

Of course, since this project does include wind, we will need to slightly modify the equations above when factoring it in. This can be accomplished by splitting the velocity of the particle into three seperate, relative velocities. The velocity of the air with respect to the ground, $\mathbf{v_{a/0}}$, the velocity of the projectile relative to the ground, $\mathbf{v_{p/0}}$, and the velocity of the projectile relative to the air, $\mathbf{v_{p/a}}$. The velocity of the air, relative to the ground is assumed to be:

$$\mathbf{v_{a/0}} = \alpha\,\hat{\mathbf{i}} + \beta\,\hat{\mathbf{j}}$$

where α and β are constant values.

These new assumptions will directly affect the drag force that the projectile experiences. The magnitude of the drag force, $\mathbf{F_D}$, becomes proportional to the speed of the projectile relative to the surrounding air, $\mathbf{v_{p/a}}$. By using similar methods to the way the first set of ODE equations were calculated, the new set of first order ODE equations governing the projectile's motion with wind is calculated to be:

$$\dot{x} = v\cos(\theta)\,\hat{\mathbf{i}} \tag{10}$$

$$\dot{y} = v\sin(\theta)\,\hat{\mathbf{j}} \tag{11}$$

$$\dot{v} = -\frac{C_d}{m}|v_{p/a}|(v - \alpha\cos(\theta) - \beta\sin(\theta)) - g\sin(\theta) \tag{12}$$

$$\dot{\theta} = -\frac{C_d}{mv}|v_{p/a}|(\alpha\sin(\theta) - \beta\cos(\theta)) - \frac{g}{v}\cos(\theta) \tag{13}$$

where $|v_{p/a}| = [(v\cos(\theta) - \alpha^2) + (v\sin(\theta) - \beta^2)]^{1/2}$ and $v = |v_{p/0}| = \dot{x}^2 + \dot{y}^2$.

2 Approach While Coding

2.1 Method

Any program that will zero in on a correct firing angle must be able to do two things: find any given trajectory and analyze it, and also run a root finding algorithm on the distance between the trajectory and the trajectory that hits a target.

To find the trajectory, Target.m calls minimum_dist.m, which uses MATLAB's built-in differential equation solver, ode45. Even without altering the default tolerance for error, this solver chooses intelligent time steps (large when the derivative is relatively constant, small when the derivative is changing rapidly) to integrate the coupled system of differential equations $\dot{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\mathbf{v}}, \dot{\theta}$ to find a trajectory for one given θ. The ODE solver stops when a local minimum for the distance between the projectile and the target is minimized (This method will be explained later) .

The function minimum_dist.m then uses the cross product between the final minimum distance vector $\mathbf{r_{t/p}}$ and the velocity vector $\mathbf{v}$ generated by ode45 to output either -1 or 1: -1 if the guess for θ is too low, 1 if the guess for θ is too high.

Target.m can then use the positive or negative output of minimum_dist.m to characterize "overshot" and "undershot" distance values and run a bisection algorithm, which begins with two (θ_1 and θ_2) guesses around a possible location of a root, then choses new guesses as it narrows a range of possible zeros of the function by cutting out ranges where the sign of a function does not change.

A brief description of bisection:

$\theta_1 = \theta_{undershot}$
$\theta_2 = \theta_{overshot}$
while $\Delta\theta > tolerance$ **do**
  $\theta_{mid} = (\theta_1 + \theta_2)/2$
  if $sign(\theta_{mid}) = sign(\theta_1)$ **then**
    $\theta_1 \leftarrow \theta_{mid}$
  else
    $\theta_2 \leftarrow \theta_{mid}$
  end if
  $\Delta\theta = \theta_1 - \theta_2$
end while

When the bisection algorithm in Target.m finds a zero location, that θ value is returned as the correct firing angle for given wind conditions and a given target.

2.2 Challenges

This project presented three major challenges to hitting a target. First, the program must be able to locate where in any given trajectory a closest approach occurs–that is, where the projectile comes the closest to hitting the target, and then stop the integrator. Second, it must be possible to decide which trajectories result from aiming to high with respect to the horizontal (overshots) and which miss the target because of angles that are too small (undershots). Third, it should be possible to find intelligent guesses for the aiming angles, both to avoid wasting computational time and ensure that the guesses for the bisection algorithm trap the root of the function.

2.2.1 Locating closest approach

In order to locate the time when the projectile is the closest to the target, our implementation uses the fact that in wind or no wind, the forces acting on the projectile do not vary in direction with respect to the coordinate axes. Gravity always acts in the $\hat{\mathbf{j}}$ direction, and the wind is some vector field with a constant vector of some $\alpha\hat{\mathbf{i}} + \beta\hat{\mathbf{j}}$ which is the same at all points. Using this, it is possible to say that when the projectile is traveling away from

the target at an increasing rate, it will never again approach the target. The magnitude of the distance vector from the projectile to the target $\mathbf{r_{t/p}}$ has reached a minimum (the point of closest approach), and the derivative of that magnitude $\frac{d\mathbf{r_{t/p}}}{dt}$ (14) passes through zero moving from negative to positive.

The distance equation:

$$\mathbf{r_{t/p}}^2 = (x - x_t)^2 + (y - y_t)^2$$

Take the derivative to get:

$$\frac{d\mathbf{r_{t/p}}}{dt} = \frac{(x - x_t)\dot{\mathbf{x}} + (y - y_t)\dot{\mathbf{y}}}{\mathbf{r_{t/p}}} \tag{14}$$

Using ode45's 'Event' option, the program locates a point where the derivative passes through zero going in the positive direction, and terminates the integration.

2.2.2 Distinguishing between undershots and overshots

Next, in order to implement bisection, it is necessary to let distance cross the axis: that is, make sure that distance has positive and negative values. The targeting program uses a cross product to characterize over and undershots.

For an overshot: The $\hat{\mathbf{k}}$ component of $\mathbf{r_{t/p(final)}} \times \mathbf{v_{(final)}} > 0$

For an undershot: The $\hat{\mathbf{k}}$ component of $\mathbf{r_{t/p(final)}} \times \mathbf{v_{(final)}} < 0$

2.2.3 Picking intelligent θ guesses

It is necessary to choose intelligent, close guesses for θ. If the guesses do not trap the root, the bisection method will fail to converge on a zero; the narrower the guesses, the more likely it is that they do not trap the root. Conversely, if the guesses are very, very wide, they run the risk of converging on a false zero for some edge cases, and the bisection algorithm will take a long time to run.

The targeting program relies on the physics of an artillery shot to make good initial θ guesses. First, know that the perturbations caused by realistic winds on the trajectory of an artillery shell will be reasonably small. By first finding a correct firing angle for the no wind condition for any given target, the guesses for θ_1 and θ_2 can be defined as slightly greater and slightly smaller than the no wind θ. Intelligent guesses for finding θ for a no wind condition to hit any target in the first quadrant are defined at the lower bound (θ_1) as the angle directly from the origin to the target, a guaranteed undershot because of gravity, while the upper bound (θ_2) is defined as $\pi/2$, the greatest possible angle for a target in the first quadrant. Solving for the accurate θ value without wind, that value is used as our initial guess for the same target with wind. The program then calculates whether that θ value results in an overshot or an undershot and sets it as θ_1 or θ_2 respectively; the other initial guess is calculated by adding or subtracting $\pi/8$ to the first guess depending on whether it

was an undershot or overshot. Making sure that we have a good range for θ means that we eliminate false minimum values at extremely wide θs, which occur because the mathematical distance function does not have simple roots.

3 Limitations

Because of the method used to pick θ guesses, this program can only hit targets in the first quadrant. Also because of our θ selections, it will only hit targets under reasonable wind conditions. If the viscous effects of the wind were stronger compared to the inertial effects of the projectile's very high velocity, strong wind might be able to perturb the true solution outside of the specified guessing range. As the projectile is, reasonable winds will not affect its ability to hit a target.

4 Efficiency

The implementation of the targeting function is made more intelligent (and efficient) by a few features of the code. The stopping criterion was the first feature, as our integrator only integrates until it reached the stop condition, i.e. where the particle begins to accelerate away from the particle. Another feature that cut down computation time was an intelligent way to determine our initial θ guesses. Our narrow range of θ guesses and the fact that we only look for the first instance of closest approach means that we often (but not always) select for the direct, and faster to compute, shots. Also, we never actually calculate the distance at a given θ, only the sign of the distance is finally calculated and returned.

In a further iterations of the code, the function could use a simplified version of the system of ODEs when there is no wind condition. By using the windless version of ODEs (excluding the alpha and beta terms) many calculations are cut out of the computation.

5 Conclusion

The method enacted during this project proved to be quite successful, allowing for any target to be hit in the first quadrant. The use of cross-products and dot-products in order to determine undershots and overshots turned out to be reasonably robust, allowing this program to calculate the correct θ, barring only extreme wind conditions.

Result: A plot of points along the integrated trajectory for a projectile fired at an angle that hits the target:

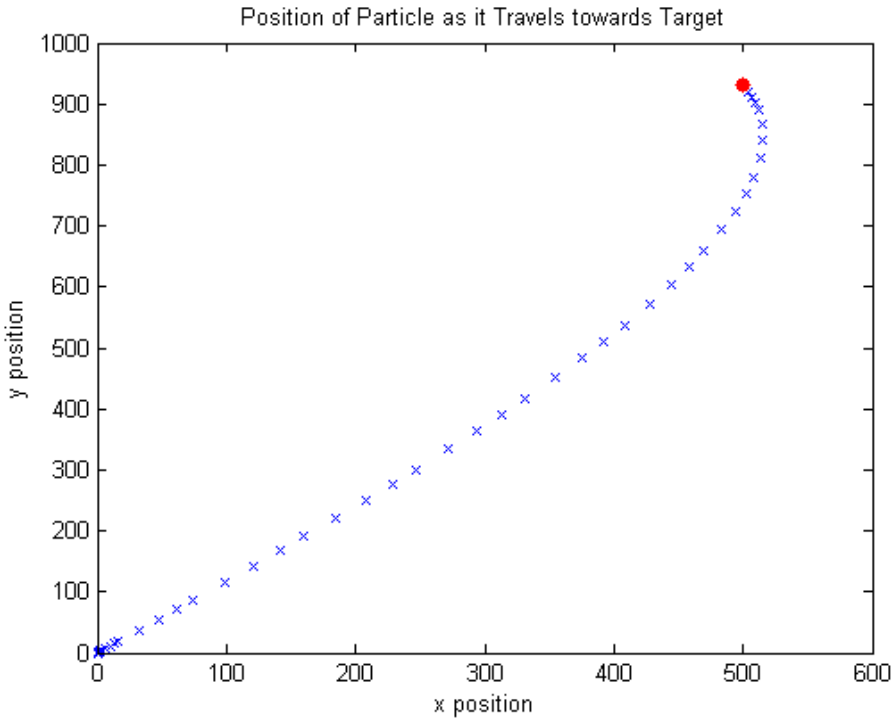

Figure 1: For a target position of (500,930) and a wind that is acting at $\mathbf{wind} = -100\hat{\mathbf{i}} + 0\hat{\mathbf{j}}$