APPM 3050 Project #01 Spring 2022

The initial hand calculations for no attractors/repulsors are due TBA

The hand calculations that include attractors/repulsors are due TBA

The final working program is due TBA

1 Overview

In this project you will be modeling the motion of a projectile fired at a target. Air resistance plays a role in this problem so the simple, exact solution does not apply. Furthermore, there will be several objects in your field of fire that can attract and/or repel your projectile at it travels to the target. The initial speed of the projectile is predetermined and your objective is to adjust the firing angle as quickly as possible to hit the target.

Although modeling projectile motion is simple enough, usually we consider the *initial value problem* of determining the projectile's path based on its initial speed and direction. Aiming the projectile to hit a designated target, on the other hand, is a *boundary value problem* where we require the solution to satisfy two spatial conditions: it must start where we are, and it must end where the target is. Since we have a choice in the initial direction, we are trying to solve an *inverse problem*: setting a parameter to result in a desired effect. Usually we deal with the *direct* or *forward problem*: what is the result of setting this parameter to some particular value?

2 The Mathematical Models

For the most simple case, the projectile's path, $\mathbf{r}(t) = x(t)\,\mathbf{i} + y(t)\,\mathbf{j}$ relative to the firing position at the origin, can be determined by solving the two-dimensional problem $\sum \mathbf{F} = m\ddot{\mathbf{r}}$ where we account only for gravitational and drag forces on the projectile. A second case will be considered in which one accounts for multiple objects in the first quadrant that can attract or repel the projectile.

2.1 No attractors or repulsors

If we write the components of the projectile's velocity $\mathbf{v}(t) = \dot{\mathbf{r}}(t) = \dot{x}(t)\,\mathbf{i} + \dot{y}(t)\,\mathbf{j}$ in polar form, then

$$\dot{x} = v\cos(\theta), \quad \dot{y} = v\sin(\theta) \tag{1}$$

where $v(t) = |\mathbf{v}| = \sqrt{\dot{x}^2 + \dot{y}^2}$ is the speed of the projectile and $\theta(t)$ is the angle of its motion measured up from horizontal, in which case comparison of the normal and tangential components of $\sum \mathbf{F} = m\ddot{\mathbf{r}}$, along the projectile's path leads to the following relations

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

where m is the mass of the projectile, and g is the acceleration due to gravity. Finally, C_D is the drag coefficient such that the drag force is proportional to the square of the speed and acts in the opposite direction of the velocity vector. The drag coefficient C_D is a measurable constant. Remember that your job is to find the projectile's location at any time, so we need to solve (1) and (2) as a system of four coupled differential equations subject to some appropriate initial conditions.

Typically when firing a weapon the initial speed (muzzle velocity), v_0, is fixed, but the firing angle is chosen by the user. Your objective is to choose a firing angle so that the projectile hits the target. If the firing angle is chosen to be θ_0, then the projectile's path is uniquely determined by the initial value problem consisting of equations (1) and (2) and the initial conditions

$$x(0) = 0, \qquad y(0) = 0, \qquad v(0) = v_0, \qquad \theta(0) = \theta_0\,. \tag{3}$$

Let's define a distance, $d(t)$, that measures the distance between the projectile and the target at any given time. We are interested in how close the projectile comes to hitting the target, in other words, we are interested in the minimum value of d, which we will call d_{min}. If we assume that d_{min} depends continuously on θ_0, then we can imagine plotting d_{min} as a function of θ_0, in which case our goal is to find the correct firing angle, θ_0^*, such that $d_{min}(\theta_0^*) = 0$.

This problem reduces to a simple root finding problem. The difficulty is that the function $d_{min}(\theta_0)$ is not at all simple — its value is determined by the solution to a system of differential equations. However, in principle d_{min} can be computed for any θ_0, thus we can apply a numerical root–finding scheme to d_{min} in order to find the optimal firing angle θ_0^*.

Since equations (1) and (2) cannot be solved by hand, we must use a numerical integration method. However, we do not really know how far to integrate. Rather than integrating to a known value of time, it is best to use a stopping criterion. If the firing angle is close to correct, then a reasonable stopping criterion is when the distance between the projectile and the target reaches a minimum. If we define the distance vector $\mathbf{d}(t) = (x(t) - x_T)\,\mathbf{i} + (y(t) - y_T)\,\mathbf{j}$, where (x_T, y_T) is the location of the target, then the distance between the projectile and the target is $d = |\mathbf{d}|$. Since the distance is minimized if and only if the square of the distance is minimized, we look for the time when $\frac{d}{dt}(d^2) = 0$. You might want to work out the details of this to arrive at a practice criterion for the stoping condition.

Another reasonable stopping criterion is that the projectile has overshot the target. Thus, you might also consider $x(t) \geq x_T$. There are several other possible stopping criterion. In any case, we can always determine d_{min} for any given trajectory resulting from a given initial firing angle θ_0.

2.2 Attractors and repulsors

Once you have your basic code working, modify it to account for several stationary objects located somewhere in the first quadrant that can attract or repel your projectile. Your projectile is outfitted with a sensor that, upon first being fired, can detect objects and determine their attractive or repulsive properties. Specifically, your sensors can measure how many objects exist, the (x, y) coordinates of each objects, the strength of the attractive or repulsive forces for each object, and of course whether they attract or repel your projectile.

You may assume that the attractive/repulsive force, $\mathbf{F}_i(t)$, of the i^{th} object on your projectile at any given time is described by

$$\mathbf{F}_i(t) = \frac{\alpha_i}{|\mathbf{r}(t) - \mathbf{r}_i|^2}\,\mathbf{n}_i(t)\,,$$

where $\mathbf{r}_i = x_i\,\mathbf{i} + y_i\,\mathbf{j}$ is the position vector of the i^{th} object relative to the origin, and $\mathbf{n}_i(t)$ is the unit vector from the current projectile position to the position of the i^{th} attractor/repulsor. If $\alpha_i > 0$ then the object attracts, while if $\alpha_i < 0$ the object repels. The magnitude of α_i determines the strength of the attraction/repulsion. In your report, you will need to clearly show how equations (1) and (2) are modified to account for the attractive and repulsive objects.

3 Coding Procedures

1. Write an integration script that uses MatLab's `ode45` integrating routine to solve (1) and (2) for some given set of initial conditions, (3), integrating over an appropriate time interval. You may hard–code the parameters $g = 9.81$ m/s^2, $C_D/m = 0.002$ m^{-1}, and $v_0 = 1500$ m/s into your program.

2. Early on in your code, you will want to call a script named `Sensors.m` that returns an array, the columns of which will contain values of $\alpha_i/m, x_i, y_i$ for each attractive/repulsive object detected. Each row of the returned array will correspond to a different attractor/repulsor. The objects will not move or change strength while your projectile is in flight so there is no need to re-call the `Sensors` file while your projectile is in flight. When we test your program, we will deposit our own `Sensors.m` file into your folder. The script `Sensors.m` has no input arguments, only output values.

3. You should seriously consider using a stopping condition with your integration scheme. Note that you will need to make sure that both your ODE definition file and your stopping criterion file have exactly the same argument list. By default, your script could integrate over an "infinite" period, but in practice integration will stop when your stopping criterion is achieved.

4. Turn your integration script into a function file that takes the target location, and the firing angle as arguments and returns the distance by which the target was missed, d_{min}. Then you can use your bisection routine to find the root of $d_{min}(\theta_0)$. Note that because distance is always non-negative, you will need to modify your returned d_{min} value such that its sign indicates whether the target was an over or undershot. This will allow your bisection method to actually find a root. You must use your own "home-brew" bisection program.

5. Finally, you will need to create a function file named `Target.m` that calls your bisection routine. This file will need to determine reasonable initial guesses for θ_0^* without any input from the user. The input values for `Target.m` will be the target coordinates, (x_T, y_T). The returned value should be the proper firing angle, θ_0 (in radians!), required to hit the target. We will call your `Target.m` file with the something similar to the following commands:

```
tic
coord = [20000, 1000];
Target(coord)
toc
```

(of course with different target coordinates) and expect that your program will return the proper firing angle to hit the target as quickly as possible.

4 Things To Consider

- You must work in groups of two or three.
- You may assume that the target is in the first quadrant.
- The location and strength of the attractors/repulsors will not change while your projectile is in flight.
- You might want to explicitly address the case where the target is in the first quadrant, but is out of range.
- We will need your answer for θ, in radians, to five significant figures with four places after the decimal. That is, $\theta =$ x.xxxx. Clearly, for the basic targets in the first quadrant, $0 \leq \theta \leq \pi/2$.
- This will be a timed competition (using `tic toc`) and extra credit will be awarded to the top five groups.
- Try to avoid performing more calculations than necessary — if a calculated value is used repeatedly, calculate it once and save the value. Pay attention to expensive calculations that are performed multiple times and make them as efficient as possible.
- How did your file `Target.m` determine your initial guesses for θ_0^* in the bisection method? Why?
- Is it possible for your stopping criterion to be met even though the projectile is not as close as it will come to the target? Will this cause a problem for your root-finding algorithm?
- Experiment with the accuracy settings for MatLab's `ode45` integrator. Justify your final choice.
- What other factors should your model consider to be truly state-of-the-art?